Integrating NoMachine with Various Authentication Methods

Prepared by: Silvia Regis

Approved by: Sarah Dryell

Last modified: 2017-01-26

Amended: A

Integrating NoMachine with Various Authentication Methods
Table of Contents

1. NoMachine Integration with Various Authentication Methods
   1.1 Document Convention and Important Notices
   1.2 Resources on the Web

2. Authentication Methods Supported
   2.1 Example Case Scenarios

3. Use SSH Key Based System Authentication on Linux
   3.1 How To Set-up SSH to Use Key Authentication
   3.2 How To Generate a SSH key-pair and Store it on the Server
   3.3 Run a Session with System Login and Key Based Authentication
   3.4 Run a Session with System Login with Key Based Authentication and SSH Key Forwarding to the Node

4. Use SSH Key Authentication with Smart Card on Linux
   4.1 How To Set-up a Smart Card Reader
   4.2 How To Set-up SSH to Use Smart Card Key Authentication
   4.3 Run a Session with System Login and Key Based Authentication and SSH Key Stored on the Smart Card
   4.4 Run a Session with System Login and Key Based Authentication, SSH Key Stored on Smart Card and SSH Key Forwarding to the Node

5. Use SSH Kerberos Authentication on Linux
   5.1 How To Install Kerberos
   5.2 How To Configure Kerberos
   5.3 How To Set-up SSH authentication using Kerberos
   5.4 How to Generate Kerberos Ticket and Forward It to Server Host
   5.5 How to Set-up PAM and Kerberos on Server Host
   5.6 How to Obtain a Ticket by Using Smart Card Authentication
5.7 Run a Session with System Login and Kerberos Ticket

5.8 Run a Session with System Login, Kerberos Ticket and Ticket Forwarding to the Node

5.9 Run a Session with System Login, Kerberos Ticket and PAM Module on the Server host

6. Use SSH Kerberos Authentication on Windows (client side)

6.1 How To Install Kerberos

6.2 How To Configure Kerberos

6.3 How To Set-up SSH authentication using Kerberos
1. NoMachine Integration with Various Authentication methods (for Connections by SSH Protocol)

Welcome to this guide about how NoMachine can work in environments where authentication methods such as SSH key-based authentication or Kerberos ticket-based authentication are enabled.

This document applies to NoMachine v. 5 or later and deals with connections by SSH protocol and NX protocol.

It is mainly intended for Administrators wishing to integrate NoMachine with their authentication infrastructure. This is not a technical guide about how to setup the system authentication infrastructure but rather a collection of How Tos based on real examples of integration made in the NoMachine labs.

Whilst the product line of server and client software is named as the Company brand, NoMachine, the software programs and libraries continue the tradition of NX naming convention. That's why this guide will use 'NoMachine Server' to refer to the product and 'NX Users', 'NX sessions' and so on to refer to the utilization of the programs.

**IMPORTANT**: NoMachine (Free) does not support connections by SSH protocol. To connect by SSH, one of the products from the NoMachine for the Enterprise range is required.

1.1. Document Convention and Important Notices

The following conventions are used in this guide:

- Case studies on Linux use Ubuntu 11.04 32 bit as operating system on client and server side. All instructions refer to such environment and may require some tuning if applied to different environments.
- Case studies on Windows use Windows XP. All instructions refer to such environment but should apply also to Windows Vista and 7.
- Commands on Linux are intended to be run from a xterm or similar as normal user or with the sudo program when specified.
- Note that Kerberos authentication is not supported on Windows server side. It's supported instead on Windows client side.

1.2. Resources on the Web

The NoMachine Web site, [http://www.nomachine.com](http://www.nomachine.com) provides a variety of online resources in conjunction with the software and its usage:

Latest News
To stay updated with the most recent releases of the NoMachine software, as well as the latest news please subscribe to the ML here: [http://www.nomachine.com/mailing-lists](http://www.nomachine.com/mailing-lists)

**The Installation Guides**
For detailed information about how to install and upgrade your NoMachine installation, please see the guides available at: [http://www.nomachine.com/all-documents](http://www.nomachine.com/all-documents)

**The Knowledge Base**
The NoMachine Knowledge Base, [https://www.nomachine.com/knowledge-base](https://www.nomachine.com/knowledge-base), provides technical documents especially for administrators as well as end-users guides, articles, and “howto's”.

**Leave Feedback About This Guide**
Our goal is to provide comprehensive and clear documentation for all NoMachine products. If you would like to send us your comments and suggestions, you can use the contact tool available at [https://www.nomachine.com/contact-request](https://www.nomachine.com/contact-request), selecting Web Quality Feedback as your option.

### 2. Authentication Methods Supported

NoMachine supports the following authentication methods:

1) For connections by NX protocol (default)

   - Password based authentication. This is the default method.
   
   - SSH key based authentication (private key).
   
   - System login with Kerberos ticket existing on client side.

2) For connections by SSH protocol

   - Login as nx user using the NoMachine server-specific RSA or DSA key (NX key) and user’s password based authentication on the system. (‘NoMachine login’ in the client GUI).
   
   - Password based authentication. ( Default ‘System login’ in the client GUI).
   
   - SSH key based authentication with private key. (‘System login’ in the client GUI).
   
   - System login with SSH key based authentication and SSH key stored on a PKCS11 smart card . (‘System login’ in the client GUI).
- System login with Kerberos ticket existing on client side. ('System login' in the client GUI).

Additionally, NoMachine supports:

**Smart cards**

NoMachine uses by default its built-in interface module to read PKCS #11 smart cards. It's also possible to configure the client to use an alternative security module. This is available only for connections by SSH and system login.

**Authentication forwarding**

An authentication agent (SSH agent) can be used to forward user's credentials inside the NoMachine session. This is available for:

- Connection by NX protocol and Kerberos ticket-base authentication
- Connection by SSH protocol and SSH key based authentication
- Connection by SSH protocol and Kerberos ticket-based authentication

**Kerberos-ticket based authentication**

Kerberos tickets can be generated on:

- i) client side
- ii) server host

i) Kerberos ticket-based authentication with tickets generated on client side is supported for both the NX and the SSH protocols. NoMachine client uses MIT Kerberos libraries by default to read Kerberos tickets. On Windows systems, Microsoft SSPI libraries are also supported.

ii) In this case, PAM is configured to obtain a ticket for password-base authentication.

**Two-factor authentication**

Two-factor authentication is supported for both the NX and the SSH protocols.

Please see this article for more technical details and examples: [http://www.nomachine.com/AR12L00828](http://www.nomachine.com/AR12L00828)

**Note for multi-node environments**
In multi-node environments user is authenticated to the node by using a key-based authentication. This uses the NoMachine server-specific RSA key ('NX key') generated at installation time, named node.localhost.id_rsa and placed in the NX/etc/keys directory under the installation directory of NoMachine server. For example on Linux: /usr/NX/etc/keys/node.localhost.id_rsa.

It is possible to customize the NoMachine RSA key-pair as explained here: https://www.nomachine.com/DT09M00103#4

### Authentication methods supported by NoMachine

<table>
<thead>
<tr>
<th>Authentication method</th>
<th>NX protocol</th>
<th>SSH protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Login with NX key + user's password</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>2. Login with user's password</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>3. Login with SSH private key (*)</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>4. Login SSH private key stored on a smart card (*)</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>5. Login with Kerberos ticket on client side (**)</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

---

(*) Support for SSH agent forwarding

(**) Support for Kerberos tickets authentication forwarding Support for two-factor authentication

### Connection settings on the end-user's device to use a specific authentication method

<table>
<thead>
<tr>
<th>Authentication method</th>
<th>Select in the client GUI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Login with NX key +</td>
<td>SSH protocol + 'Use the</td>
</tr>
</tbody>
</table>
## Integrating NoMachine with Various Authentication Methods

<table>
<thead>
<tr>
<th></th>
<th>Method Description</th>
<th>Authentication Options</th>
</tr>
</thead>
</table>
| 2 | Login with user's password | SSH protocol + 'Use the system login' + Password  
NX protocol + 'Password' |
| 3 | Login with SSH private key (*) | SSH protocol + 'Use the system login' + 'Private key'  
NX protocol + 'Private key' |
| 4 | Login SSH private key stored on a smart card (*) | SSH protocol + 'Use the system login' + 'Smart card' |
| 5 | Login with Kerberos ticket on client side (**) | SSH protocol + 'Kerberos'  
NX protocol + 'Kerberos' |

*(*) **Support for SSH agent forwarding**

(***) **Support for Kerberos tickets authentication forwarding**

**Support for two-factor authentication**

No settings needed on client side, it's a server side configuration
Some examples of how to configure the environment

1. Connection by NX or SSH protocol: SSH key based authentication (private key)

   How To
   - Set-up SSH to use key authentication.
   - Generate SSH key-pair and store it on the server.

2. Connection by SSH protocol: SSH key based authentication with private key stored on a smart card

   How To
   - Set-up a smart card reader, generate smart card keys and store them on smart card.
   - Set-up SSH to use smart card key authentication.

3. Connection by NX or SSH protocol: Kerberos ticket-based authentication

   Kerberos ticket available on the client
   PAM Kerberos Module on NoMachine Server host

   How To
   - Set-up SSH authentication using Kerberos.

2.1. Example Case Scenarios

Scenarios have been deployed in the following environments:

- Client OS (and Kerberos Client OS): Ubuntu 11.04 32 bit and Windows XP.
- Kerberos version: Kerberos 5 release 1.8.3 for Linux and Kerberos 5 release 1.9.2 for Windows.
- PKCS#15 compliant smart card to be compatible with OpenSC.

Client is the machine from which user is connecting, namely the NoMachine Enterprise Client host. Server is the system where user wants to connect to, namely the machine where NoMachine Server (and Kerberos) or Node is installed.

While steps for configuring the environment are universal, specific instructions might be needed to adapt to a different environment. For example names of packages or paths may depend on the operating system.
3. Use SSH Key Based System Authentication on Linux

3.1. How To Set-up SSH to Use Key Authentication

To be able to use key based authentication in SSH connections, you need to ensure that key authentication is not disabled explicitly.

On the client machine verify the SSH configuration files placed in: /etc/ssh/ssh_config or in: ~/.ssh/config in case of a per-user settings.

The following entry should not be present or should be commented-out:

```
PubkeyAuthentication no
```

Then verify that this entry is not active in the SSHD configuration file, /etc/ssh/sshd_conf, on the NX Server host.

Now everything should be ready to use key authentication: SSH on default settings will try to use your public key before it will prompt for your password.

3.2. How To Generate a SSH Key-pair and Store it on the Server

Let's assume that you are logged on your client machine as user nxtest.

**Step 1** - Generate the SSH key-pair by using the SSH key generator utility:

```
$ ssh-keygen
```

If no option is specified, it generates a couple of RSA keys. Otherwise, use the option `-t` to specify type of key to create.

**Step 2** - Name the private key.

You will be prompted to enter a file name for the private key. Public will be stored as `<private_key_file_name>.pub`. If you leave it blank, keys will be created in the default folder, i.e. the one shown in brackets:

```
Enter file in which to save the key (/home/nxtest/.ssh/id_rsa):
```

**Step 3** - Give a password.

In the next step you will be prompted for a password. You can leave it blank, but it is recommended to enter a password:

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```
Step 4 - Put the keys in the appropriate place.

Now that you have your own key-pair, put them in appropriate places for SSH to find them when needed.

By default, the private key is already stored on your client computer in the right place. If you are logged as nxtest user, that key is at: /home/nxtest/.ssh/id_rsa. Just make sure that nobody but you will have access to it. If you saved the key in another place, move it to /home/nxtest/.ssh.

The public key, on the other hand, should be distributed to every server you wish to have access to. So if you want to have access as user <username> on server <hostname> you should copy the content of your id_rsa.pub file and paste it in the following file on <hostname>:

```
/home/<username>/.ssh/authorized_keys
```

Now each time you connect via ssh to <hostname>:

```
$ ssh <username>@<hostname>
```

You will be authenticated using your public key, and prompted for the passwordphrase to it, if you specified a password during the key generation.

TIP

If you have chosen a different path from the default one for storing the private key (id_rsa), please remember to run the ssh command with the '-i' option to specify a file from which the private key for authentication is read:

```
ssh <username>@<hostname> -i <id_rsa_file_path>
```

3.3. Run a NoMachine Session with System Login and Key Based Authentication

Pre-requisite

You have generated your own SSH key-pair and placed the public key on the desired server.

Run the NoMachine client on your device:

- Create a new session or select an existing one. Open its Connection panel by double click on the session item.
- Then click on the Edit button.
- Choose Protocol SSH and click on the Advanced button.
- Choose 'Use the system login' and click on the Settings button.
• Select 'Private key' as authentication method and load your private SSH key file. The client should load the private key if it is placed in a default location. Otherwise, specify path to your private key.
• Click on Continue button until you reach the initial Connection panel.
• Click on Connect button to run the session.
• If you have generated the key-pair with a password, the NoMachine client will prompt you for it.

3.4. Run a Session with System Login with Key Based Authentication and SSH Key Forwarding to the Node

Pre-requisite
You have generated your own SSH key-pair and placed the public key on the desired server.

Run NoMachine client and follow steps described in the previous paragraph.

Additionally, select the 'Forward authentication' checkbox in the System login settings panel.

4. Use SSH Key Authentication with Smart Card on Linux

Support for authentication with smart card has been set-up by relying on the Public Key Infrastructure (PKI) and using an OpenSC compatible smart card.

4.1. How To Set-up a Smart Card Reader

Pre-requisite
You have installed a set of tools and libraries necessary to deal with smart card:

- opensc (http://www.opensc-project.org/opensc)
- pcsclite (http://pcsclite.alioth.debian.org)
- engine_pkcs11 (http://www.opensc-project.org/engine_pkcs11)

Step 1 - Erase the smart card:

$ pkcs15-init -E

This command will erase the card prior to creating the PKCS #15 data structure.
Step 2 - Create the PKCS #15 structure.

Now you can create the PKCS #15 structure by running the following command with options at your convenience, for example:

```
$ pkcs15-init --create-pkcs15 --profile pkcs15+onepin --use-default-transport-key --pin 0000 --puk 111111 --label "NX Test 01"
```

Verify that the command has been executed properly:

```
$ pkcs15-tool --dump
```

You should obtain an output similar to:

```
PKCS#15 Card [NX Test 01]:
Version : 1
   Serial number : 3035350110260511
Manufacturer ID: EnterSafe
Last update : 20111006144034Z
Flags : EID compliant

PIN [User PIN]
Com. Flags: 0x3
   ID : 01
   Flags : [0x32], local, initialized, needs-padding
   Length : min_len:4, max_len:16, stored_len:16
   Pad char : 0x00
   Reference : 1
   Type : ascii-numeric
   Path : 3f005015

Step 3: Generate the RSA keys.

To generate the RSA keys you can use the following command and specify options suitable for you. For example:

```
$ pkcs15-init --generate-key rsa/2048 --auth-id 01 --pin 0000
```

Verify that command above has been executed properly:

```
$ pkcs15-tool --list-keys
```

Output of this command should be similar to:

```
Private RSA Key [Private Key]
Object Flags : [0x3], private, modifiable
   Usage : [0x4], sign
   Access Flags : [0x1D], sensitive, alwaysSensitive, neverExtract, local
   ModLength : 2048
```
Key ref : 1 (0x1)
Native : yes
Path : 3f005015
Auth ID : 01
ID : 19d22905b8c4c766240f697ea81c0fe0750cb928
GUID : {19d22905-b8c4-c766-240f-697ea81c0fe0}

TIP
Remember ID from output of command above, you will need it later.

Step 4 - Check the available slots:

$ pkcs11-tool --list-slots --module /usr/local/lib/opensc-pkcs11.so

Output of this command should be similar to:

Available slots:
Slot 0 (0xffffffff): Virtual hotplug slot (empty)
Slot 1 (0x1): Feitian SCR310 00 00
token label: NX Test 01 (User PIN)
token manuf: EnterSafe
token model: PKCS#15
token flags: rng, login required, PIN initialized, token initialized
serial num : 3035350110260511

TIP
Remember Slot number from output of command above, you will need it later.

Step 5 - Generate the OpenSSL certificate.

Run the OpenSSL command line tool:

$ openssl

and load ENGINE, the cryptographic module support engine. Remember to set a proper path to engine. For example:

OpenSSL> engine dynamic -pre SO_PATH:/usr/lib/engines/engine_pkcs11.so
-pre ID:pkcs11 -pre LIST_ADD:1 -pre LOAD -pre
MODULE_PATH:opensc-pkcs11.so
If you are not using Certificate Authority (e.g. Kerberos), require self-signed certificate:

```
OpenSSL> req -engine pkcs11 -new -key slot_SLOT-id_ID -keyform engine -x509 -out cert.pem -text
```

Otherwise, if you are using Kerberos, generate request for certificate:

```
OpenSSL> req -engine pkcs11 -new -key slot_<SLOT>-id_<ID> -keyform engine -out client.req
```

<SLOT> is the Slot number retrieved from output of command at Step 4 and <ID> is value of field ID retrieved from output of command at Step 3.

TIP
Additional steps are necessary to use smart cards with Kerberos. Please consult section of this guide dedicated to Kerberos.

**Step 6 - Store the certificate:**

```
$ pkcs15-init --store-certificate cert.pem --auth-id 01 --id <ID> --format pem
```

<ID> should be replaced with values from Step 4.

TIP
If you are using Kerberos, cert.pem should be generated using CA of Kerberos KDC (Key Distribution Center). This is explained in the How To available in the section of this guide dedicated to Kerberos.

## 4.2. How To Set-up SSH to Use Smart Card Key Authentication

**Pre-requisite**
You have installed set of tools and libraries necessary to deal with smart card:

- opensc ([http://www.opensc-project.org/opensc](http://www.opensc-project.org/opensc))
- pcsclite ([http://pcsclite.alioth.debian.org](http://pcsclite.alioth.debian.org))
- engine_pkcs11 ([http://www.opensc-project.org/engine_pkcs11](http://www.opensc-project.org/engine_pkcs11))

**Step 1 - Retrieve the SSH-RSA public key from your smart card**
$ pkcs15-tool --list-public-keys

This command lists the data structures on a smart card. Its output is similar to:

Public RSA Key [Private Key]
Com. Flags : 2
Usage : [0x4], sign
Access Flags: [0x0]
ModLength : 2048
Key ref : 0
Native : no
Path : 3f0050153000
Auth ID :
ID : 19d22905b8c4766240f697ea81c0fe0750cb928

Then retrieve the RSA public part of the key by running:

$ pkcs15-tool --read-ssh-key <ID>

where <ID> is obtained from output 'pkcs15-tool --list-public-keys' command.

The command above reads the public key with the given ID and writes it to the output in format suitable for $HOME/.ssh/authorized_keys file.

Now you need to put the public key on the server host. Copy that output and paste it on the server host in your home, namely in the ' ~/.ssh/authorized_keys' file.

SSHd on the server host should now be ready to authenticate using the SSH keys obtained from your smart card.

TROUBLESHOOTING
If the smart card is not inserted into the reader, SSH will not prompt to insert it. SSH will just print "no slots" and try to use password authentication.

Step 2 - Enabling SSH Key forwarding.
This step is necessary only if you need to forward the SSH key obtained from the smartcard to the server host.

On your server host

First of all, check if the SSH authentication program, ssh-agent, is running. To do that, you can verify if the proper environmental variable is set by running from xterm or similar:

$ echo $SSH_AUTH_SOCK

If the result of the command is an empty string, run:

$ eval `ssh-agent`
If ssh-agent is running, you should get an output similar to:

Agent pid 9256

Then, add the SSH public key to the ssh-agent:

$ ssh-add -s /usr/local/lib/opensc-pkcs11.so

(note that path to opensc-pkcs11.so can be different, you may use 'find / -name opensc-pkcs11.so' to retrieve it.

and verify if the key has been added successfully:

$ ssh-add -l

Output of this command should be similar to:


Finally, enable agent forwarding, if it's not already active. Check the SSHD configuration file, namely '/etc/ssh/sshd_config' file to verify that it has the following entry active:

AllowAgentForwarding yes

On your client host

Enable agent forwarding by editing the SSH configuration file, namely ~/.ssh/config. Make sure that this entry exists and is not commented-out:

ForwardAgent yes

Alternatively you can run ssh command with '-A' parameter which enables forwarding of the authentication agent connection.

4.3. Run a Session with System Login and Key Based Authentication and SSH Key Stored on a Smart Card

Pre-requisite

You have generated your own SSH public - private key-pair and placed the public key on the desired server host.
Run NoMachine client on your device:

- Create a new session or select an existing one. Open its 'Connection' panel by double click on the session item.
- Then click on the Edit button.
- Choose Protocol SSH and click on the Advanced button.
- Choose 'Use the system login' and click on the Settings button.
- Select 'Smart card' as authentication method. The client will use a SmartCart authentication library shipped with NoMachine by default. Alternatively, you can specify path to your own library by choosing 'Use an alternate security module'.
- Click on Continue button until you reach the initial Connection panel.
- Click on Connect button to run the session.
- Enter your PIN when you will be prompted for.

The session should now be up and running.

### 4.4. Run a Session with System Login and Key Based Authentication, SSH Key Stored on Smart Card and SSH Key Forwarding to the Node

**Pre-requisite**

You have generated your own SSH public - private key-pair and placed the public key on the desired server host.

You need to follow steps described in the previous paragraph.

Additionally, select the 'Forward authentication' checkbox in the System login settings panel.

### 5. Use SSH Kerberos Authentication on Linux

#### 5.1. How To Install Kerberos

You need to install and configure Kerberos on both the client and the server host.

**Install Kerberos on server host**

**Step 1** - Install krb5-kdc and krb5-admin-server packages:

```
$ sudo apt-get install krb5-kdc krb5-admin-server
```

During installation of those packages you will be prompted to enter the following data:
REALM_NAME, KDC Host Name, Admin Server Host Name. Specify them at your convenience.

**Step 2** - Create new realm (or administrative domain):

```
$ sudo krb5_newrealm
```

*Install Kerberos on client host*

**Step 1** - Install krb5-user package:

```
$ sudo apt-get install krb5-user
```

### 5.2. How To Configure Kerberos

*Configure Kerberos on server host*

**Step 1** - Add new principal (or entry) to Kerberos KDC (Key Distribution Center) database:

```
$ sudo kadmin.local
kadmin.local: addprinc <username>
```

**TIP**

If you want to use Kerberos with SSH or PAM, `<username>` should be one of the existing system users on that server machine. Unless you're using PAM and `~/.k5login` file. Then you can use any names and add them to `~/.k5login` file.

**Step 2** - Add a principal to KDC for each client machine:

```
$ sudo kadmin.local -q "addprinc -randkey host/<fully_qualified_hostname>"
$ sudo kadmin.local -q "ktadd host/<fully_qualified_hostname>"
```

### 5.3. How To Set-up SSH authentication using Kerberos

SSH supports 3 ways to authenticate using Kerberos:

- Use pre-obtained ticket generated by using the `kinit` command.
- Set up PAM (Pluggable Authentication Modules) on server host to deal with Kerberos and create ticket on the server host.
- Obtain a ticket by using smart card authentication.

### 5.4. How to Generate Kerberos Ticket and Forward It to Server Host
Step 1 - Obtain the Kerberos ticket for the default principal.

On your client host, run:

```
$ kinit
```

You will be prompted to enter your Kerberos password.

If you want to forward this ticket automatically to the server host system you want to access, you can modify the `/etc/krb5.conf` file and ensure that the following entry is present in the `[appdefaults]` section:

```
forwardable = true
```

Otherwise you can just add the `-f` option to `kinit` command.

To check if the ticket is forwardable, you can run:

```
$ klist -f
```

and verify if 'F' is present among the flags.

Step 2 - Configure SSH to use GSSAPI authentication.

Now you have to configure SSH to use GSSAPI authentication and, if needed, forward the ticket.

You can edit the global ssh settings file located in `~/etc/ssh/ssh_config` on client machine and ensure that the following settings are not commented out. They have to be set as it follows:

```
GSSAPIAuthentication yes
GSSAPIDelegateCredentials yes
```

This last entry is necessary to forward the ticket and can be omitted if you don't need it.

If you prefer to apply changes on a per-user basis, edit the per-user settings in the `~/.ssh/config` file.

As an alternative, you can add the `-K` option when running the `ssh` command: it enables GSSAPI-based authentication and forwarding of GSSAPI credentials to the server.

**TROUBLESHOOTING**

If tickets are not forwarded due to a pre-authentication issue, you can solve it by explicitly disabling pre-authentication while setting up Kerberos principals.

You can do that by running the following commands:

```
$ sudo kadmin.local
kadmin.local: modprinc -requires_preauth krbtgt/<REALM_NAME>
```
5.5. How to Set-up PAM and Kerberos on Server Host

**Step 1** - Install the PAM module for Kerberos on the server host:

```bash
$ sudo apt-get install libpam-krb5
```

The PAM module should work out-of-the-box with default settings since the installation sets `pam-krb5` module as the first one to be tried while authenticating.

If you need to change this behavior or add some custom options, you can modify the `/etc/pam.d/common-auth` file.

Now, when you log-in to the server host with PAM configured to use the kerberos module, you have to enter you Kerberos password. You will obtain the kerberos ticket on the server host.
5.6. How to Obtain a Ticket by Using Smart Card Authentication

Let's assume that you want to be your own Certificate Authority (CA) and you want to create and sign your own certificate requests.

Step 1 - Create the CA private key:

$ openssl genrsa -out cakey.pem 2048

Step 2 - Generate the CA certificate:

$ openssl req -key cakey.pem -new -x509 -out cacert.pem

Step 3 - Generate the KDC (key distribution center) key:

$ openssl genrsa -out kdckey.pem 2048

Step 4 - Generate certificate request for KDC:

$ openssl req -new -out kdc.req -key kdckey.pem

Step 5 - Create OpenSSL extension file.

This file should be a regular text file containing content reported between ---CUT--- section below. Create such file in the current directory and give it any name. We will refer to it later as to <extension_file_name>.

--- CUT ---

[ kdc_cert ]
basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
keyUsage = nonRepudiation, digitalSignature, keyEncipherment,
keyAgreement

#Pkeinit EKU
extendedKeyUsage = 1.3.6.1.5.2.3.5

subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# Copy subject details

issuerAltName=issuer:copy

# Add id-pkinit-san (pkinit subjectAlternativeName)
subjectAltName=otherName:1.3.6.1.5.2.2;SEQUENCE:kdc_princ_name

[kdc_princ_name]
realm = EXP:0, GeneralString:${ENV::REALM}
principal_name = EXP:1, SEQUENCE:kdc_principal_seq

[kdc_principal_seq]
name_type = EXP:0, INTEGER:1
name_string = EXP:1, SEQUENCE:kdc_principals

[kdc_principals]
princ1 = GeneralString:krbtgt
princ2 = GeneralString:${ENV::REALM}

[ client_cert ]

# These extensions are added when 'ca' signs a request.

basicConstraints=CA:FALSE
keyUsage = digitalSignature, keyEncipherment, keyAgreement
extendedKeyUsage = 1.3.6.1.5.2.3.4
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

subjectAltName=otherName:1.3.6.1.5.2.2;SEQUENCE:princ_name

# Copy subject details

issuerAltName=issuer:copy

[princ_name]
realm = EXP:0, GeneralString:${ENV::REALM}
principal_name = EXP:1, SEQUENCE:principal_seq

[principal_seq]
name_type = EXP:0, INTEGER:1
name_string = EXP:1, SEQUENCE:principals

[principals]
princ1 = GeneralString:${ENV::CLIENT}

-- CUT ---
**Step 6** - Generate proper certificate for KDC:

```
$ REALM=<REALM_NAME>; export REALM
$ CLIENT=<principal_name>; export CLIENT
```

Replace `<REALM_NAME>` and `<principal_name>` with the proper realm and principal.

```
$ openssl x509 -req -in kdc.req -CAkey cakey.pem -CA cacert.pem -out kdc.pem -extfile <extension_file_name> -extensions kdc_cert -CAcreateserial
```

**Step 7** - Generate client certificate:

```
$ openssl genrsa -out clientkey.pem 2048
```

**Step 8** - Generate certificate request:

```
$ openssl req -new -key clientkey.pem -out client.req
```

<table>
<thead>
<tr>
<th>TIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>When using smart cards, please execute steps 7 and 8 according to instructions reported in paragraph: 4.1. How To Set-up a Smart Card Reader</td>
</tr>
</tbody>
</table>

**Step 9** - Generate certificate for the request:

```
$ REALM=<REALM_NAME>; export REALM
$ CLIENT=<user_name>; export CLIENT

$ openssl x509 -CAkey cakey.pem -CA cacert.pem -req -in client.req -extensions client_cert -extfile <extension_file_name> -out client.pem
```

**Step 10** - KDC configuration in ‘/etc/krb5.conf’ file on the server host should be:

```
pkinit_identity = FILE:/var/lib/krb5kdc/kdc.pem, FILE:/var/lib/krb5kdc/kdckey.pem
pkinit_anchors =FILE:/var/lib/krb5kdc/cacert.pem
```

<table>
<thead>
<tr>
<th>TIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you want users to authenticate with key obtained by pkinit (e.g. smartcard) and not with password, modify principals to use pre_auth:</td>
</tr>
</tbody>
</table>

```
$ sudo kadmin.local -q "modprinc +requires_preauth <principal_name>"
```

**Step 11** - Client configuration (in ‘/etc/krb5.conf’ file) on the client host should be:
[realms]
<REALM_NAME> = {
    pkinit_anchors = FILE:/etc/krb5/cacert.pem
    pkinit_identities = FILE:/etc/krb5/client.pem,/etc/krb5/clientkey.pem
}

TIP

- KDC has 'pkinit_identities' and client has: 'pkinit_identities' option.
- cacert.pem should verify client key, so be sure to use the proper one.
- When using smartcard keys, use: 'pkinit_identities = PKCS11:/usr/local/lib/opensc-pkcs11.so'

5.7. Run a Session with System Login and Kerberos Ticket

Pre-requisite

You have installed and configured Kerberos properly on both server and client host.

To be sure you have a valid Kerberos ticket, first of all run:

$ klist

Output of this command should be similar to this:

Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: nxuser@NOMACHINE.COM

Valid starting Expires Service principal
10/07/11 09:31:16 10/07/11 19:31:16 krbtgt/NOMACHINE.COM@NOMACHINE.COM
renew until 10/08/11 09:31:17

If you don't have yet a valid ticket, obtain it by running:

$ kinit

Enter your Kerberos password when prompted and check once again if ticket was obtained successfully.

Once you are sure you have a valid ticket, run NoMachine client on your device:
• Create a new session or select an existing one. Open its 'Connection' panel by double click on the session item.

• Then click on the Edit button.
• Choose Protocol SSH and click on the Advanced button.
• Choose 'Use the system login' and click on the Settings button.
• Select any of the Kerberos options as authentication method.

• Click on Continue button until you reach the initial Connection panel.
• Click on Connect button to run the session.

• You should be prompted for your username.

5.8. Run a Session with System Login, Kerberos Ticket and Ticket Forwarding to the Node

Pre-requisite

• You need to have installed and configured Kerberos properly on server host.
• You need to get NoMachine server authenticated to the Node and add it as node to the server regardless if it's localhost or not. Localhost is not mapped properly and therefore not recognized by Kerberos. To do that you can use tools built in nxnode and nxserver and run the following commands:

  # ./nxnode --keyadd /usr/NX/etc/keys/node.localhost.id_dsa.pub
  # ./nxserver --nodeadd <name of the node>

Note that this works only if your NX Server supports multi-node.

Check if you have valid and forwardable Kerberos ticket:

$ klist -f

The output of this command should be similar to this:

Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: nxuser@NOMACHINE.COM

Valid starting Expires Service principal
10/07/11 09:31:16 10/07/11 19:31:16 krbtgt/NOMACHINE.COM@NOMACHINE.COM
renew until 10/08/11 09:31:17, Flags: FPRi

Check if 'F' is present among the flags. This means that the ticket is forwardable. If you don't have valid ticket, obtain it by running:

$ kinit -f
Enter your Kerberos password when prompted and check if ticket was obtained successfully.

**TIP**

If you have modified the `/etc/krb5.conf` file to contain `forwardable = true` in `[appdefaults]` section, then you don't need to use the `-f` option.

## 5.9. Run a Session with System Login, Kerberos Ticket and PAM Module on Server host

### Pre-requisites

- You need to have installed and configured Kerberos properly on server host.
- You need to get NoMachine Server authenticated to the Node and add it as node to the server regardless if it's localhost or not. localhost is not mapped properly and therefore not recognized by Kerberos. To do that you can use tools built in nxnode and nxserver and run the following commands:

  ```
  # ./nxnode --keyadd /usr/NX/etc/keys/node.localhost.id_dsa.pub
  # ./nxserver --nodeadd <name of the node>
  ```

  Note that this works only if your NX Server supports multi-node.

Run NoMachine client on your device:

- Create a new session or select an existing one. Open its 'Connection' panel.
- Choose 'Use the system login'.

Exit the configuration panels and launch the session. You should be prompted for your login and password. Enter your Kerberos password, not your system password.

If everything has been done correctly, your Kerberos credentials should be available in the session. To check if the ticket has been properly forwarded, run an xterm or similar into your session and execute:

```
$ klist
```

Output of this command should be similar to:

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: nxuser@NOMACHINE.COM
```
6. Use SSH Kerberos Authentication on Windows (Client Side)

6.1. How To Install Kerberos

You need to install and configure Kerberos on the client host:

- Download the installer of MIT Kerberos for Windows from: http://web.mit.edu/kerberos/dist/index.html# krb5-1.9
- Run the installer with administrative privileges.

6.2. How To Configure Kerberos

Configure Kerberos on client host by editing manually the krb5.ini file

**Step 1** - Open the krb5.ini file with administrative privileges.

To see where the krb5.ini file is located, please refer to the official documentation. The announce of Kerberos for Windows Release 4.0.1 reports:

"The krb5.ini configuration file is no longer installed in C:\Windows. Instead, it is installed in \CSIDL_COMMON_APPDATA, which is C:\ProgramData\MIT\Kerberos5 on systems newer than Windows XP, where this location translates to C:\Documents and Settings\All Users\Application Data." (http://web.mit.edu/kerberos/kfw-4.0/kfw-4.0.html)

**Step 2** - Edit the file as it follows:

- Add new domain in [domain_realms], e.g:

  nomachine.com = "NOMACHINE.COM"

- Add new realm in [realms], e.g:

  NOMACHINE.COM = {
  admin_server = "nomachine.com"
  kdc = "nomachine.com"
master_kdc = "nomachine.com"
}

- Optionally, set default_realm in [libdefaults], e.g:

  default_realm = "NOMACHINE.COM"

As an alternative, you may use the GUI tool shipped with the Kerberos package:

**Step 1** - Run the netidmgr.exe tool with administrative privileges.

**Step 2** - Choose Options -> General -> Realms from the menu and apply the same settings as described above for editing manually the krb5.ini file.

### 6.3. How To Set-up SSH Authentication Using Kerberos

**Step 1** - Obtain a Kerberos ticket from KDC (Key Distribution Center):

To obtain the Kerberos ticket, run the netidmgr.exe tool shipped with the Kerberos package and choose Credentials -> Obtain new credentials from the menu.

**Step 2** - Configure SSH to use GSSAPI authentication and, if needed, forward the ticket:

Edit the ssh settings file and add the following parameters to ssh argument:

- `o GSSAPIAuthentication=yes`
- `o GSSAPIDelegateCredentials=yes`

This last entry is necessary to forward the ticket and can be omitted if you don't need it.

For example:

```
ssh -o GSSAPIAuthentication=yes localhost
```

As an alternative, you can add the `-K` option when running the 'ssh' command: it enables GSSAPI-based authentication and forwarding of GSSAPI credentials to the server.

**TIP**

Remember to use the proper fully qualified domain name (FQDN) of your KDC server for SSH connection.

Alternatively, you need to add entry for KDC to hosts files. On Windows systems hosts file should be placed in:

```
<WINDIR>/system32/drivers/etc/hosts
```

For example, you hosts file could be similar to:
127.0.0.1 localhost
89.72.9.222 testdrive.nomachine.com
TROUBLESHOOTING

If SSH connection fails with an error similar to: "Unspecified GSS failure. Minor code may provide more information. Incorrect net address". try to use an addressless ticket. To do that, run the netidmgr.exe tool and check the addressless option in the New credentials window. Then retrieve the new ticket.