NOMACHINE	Integrating NoMachine with Various Authentication Methods	
Prepared by: Silvia Regis	**N°**: D-710_004-NTG-NMC	
Approved by: Sarah Dryell	**Last modified**: 2020-08-07	**Amended**: A

Integrating NoMachine with Various Authentication Methods

Table of Contents

Introduction

1. NoMachine Integration with Various Authentication Methods

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

This document 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.

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.

See chapter 2, Authentication Methods Supported to verify which authentication methods are supported for the NX and the SSH protocol.

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, 7 and later.
- Commands on Linux are intended to be run from a xterm or similar as normal user or with the sudo program when specified.

TIP

For connections by **SSH protocol**: Kerberos authentication is not supported on Windows server side.It's supported instead on Windows client side.
Kerberos authentication on Windows is fully supported for connections by **NX protocol**.

1.2. Resources on the Web

Online Resources
Visit the NoMachine Support Area to access a variety of online resources included the NoMachine Forums, tutorials and FAQs: https://www.nomachine.com/support

Find a list of all documents and tutorials: https://www.nomachine.com/all-documents

Use the Knowledge Base search engine to access articles, FAQs and self-help information: https://www.nomachine.com/knowledge-base

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, selecting Web Quality Feedback as your option.

2. Authentication Methods Supported

NoMachine supports the following authentication methods:

I **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.

II **For connections by SSH protocol**
- Password based authentication.
- SSH key based authentication (private key).
- SSH key based authentication with a key provided by SSH agent (available since v. 6.3.6)
- SSH key based authentication and SSH key stored on a PKCS11 smart card.
- System login with Kerberos ticket existing on client side.

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 or on
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. Kerberos tickets generated on server side are supported on Windows only for connections by NX protocol. They are not supported when connecting by SSH protocol.

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:
https://www.nomachine.com/AR12L00828

TIP

Multi-node environments (Enterprise Terminal Server + Terminal Server Nodes)
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.

2.1. Comparison Tables

Authentication methods supported

Authentication method	NX protocol	SSH protocol
Login with user's password	yes	yes
Login with SSH private key	yes	yes
Login with SSH private key provided by SSH agent (available since v. 6.3.6)	-	yes
Login with SSH private key stored on a smart card	-	yes
Login with Kerberos ticket on client side	yes	yes
Support for SSH agent forwarding	no	yes
Support for Kerberos tickets authentication forwarding	yes	yes
Support for two-factor authentication	yes	yes

Authentication methods and connection setting in the client GUI

To edit an existing connection, right mouse click on the item listed in 'Recent connection', then click on 'Advanced' to choose which authentication method you want to use. Click on 'Settings' to provide the private-key or select 'Forward authentication' for those methods supporting it.

Authentication method	Select in the client GUI
Login with user's password	'Password'
Login with SSH private key	'Private key'
Login with SSH private key provided by SSH agent	'Authentication agent'
Login with SSH private key stored on a smart card	'Smart card'
Login with Kerberos ticket on client side	'Kerberos'
Support for SSH agent forwarding	'Private key' + 'Forward authentication' 'Authentication agent' + 'Forward authentication' 'Smart card' + 'Forward authentication'
Support for Kerberos tickets authentication forwarding	'Kerberos' + 'Forward authentication'
Support for two-factor authentication	No settings needed on client side, it's a server side configuration

2.2. Use-Cases

This guide provides some step-by-step examples to configure the environment for :

I SSH key based authentication (private key)
II SSH key based authentication with private key stored on a smart card (only for SSH protocol)
III Kerberos ticket-based authentication

Use-cases have been deployed in the following environments:
Client OS (and Kerberos Client OS) : Ubuntu 11.04 32 bit and Windows XP.
Server OS (and Kerberos Server OS): Ubuntu 11.04 32 bit. 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.

Use-cases have been re-tested with NoMachine v. 6.3.6 on:
Client OS (and Kerberos Client OS) : Ubuntu 18.04 64 bit and Windows 10.
Server OS (and Kerberos Server OS): Ubuntu 18.04 64 bit. Kerberos version: krb5-user 1.16-2build1 for Linux and Kerberos Release 4.1 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 NoMachine Terminal Server 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.

TIP

For Kerberos ticket-based authentication
Ensure that the time is correctly synchronized between client and Kerberos Key Distribution Center.

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 Authentication

Let's assume that you are logged on your (local) 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 fo RSA keys. Otherwise, use the option '-t' to specify type of key to create.

Note that the new OpenSSH format introduced by openssh version 7.8p1-1 is not supported yet. With the new format, the private key has "OPENSSH" written in the first line instead than "RSA". As a possible workaround, run ssh-keygen with the -m PEM option which uses the old format:

```
$ ssh-keygen -m PEM
```

If you're on Windows and use PuTTYgen (https://www.puttygen.com), be sure keys are in the OpenSSH format or convert it. To do that choose the key file in the PuTTYgen main window. Then go to Conversions->Export OpenSSH key to export your private key and save it.

Step 2 - Name the private key.
You will be prompted to enter a file name for the private key. Public part 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, e.g.:

```
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 on server 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 passphrase 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. How to Configure the Client to Use a Private Key

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. Click on its icon to open the context menu and click on 'Edit connection'.
-Choose Protocol SSH and click on the Advanced button.
-Select 'Private key' and open its Settings. -The client should load the private key if it is placed in a default location. Otherwise, specify path to your private key.
-Confirm your settings and connect. If you have generated the key-pair with a password, the NoMachine client will prompt you for it.

TIP

Select the 'Forward authentication' checkbox in the Private key's settings panel if you want to forward credentials into the session.

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 hhttps://github.com/OpenSC/OpenSC/wiki
pcsclite https://pcsclite.apdu.fr
engine_pkcs11 (which is part of libp11 library) https://github.com/OpenSC/libp11

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, you obtain an output similar to the following:

```
$ pkcs15-tool --dump

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, you should get an output similar to the following:

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

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.
Run the following, output should be similar to this example:

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

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

```
$ openssl

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 https://github.com/OpenSC/OpenSC/wiki
pcsclite https://pcsclite.apdu.fr
engine_pkcs11 https://github.com/OpenSC/libp11

Step 1 - Retrieve the SSH-RSA public key from your smart card
This command lists the data structures on a smart card. It's output is similar to:

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

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 : 19d22905b8c4c766240f697ea81c0fe0750cb928
```

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.

Then 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 the following command. If ssh-agent is running, you should get an output similar to:

```
$ eval `ssh-agent`

$ Agent pid 9256
```

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

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

Verify if the key has been added successfully, output should be similar to:

```
$ ssh-add -l
```

```
2048 86:39:70:ef:0d:11:d6:2b:9c:1c:e2:06:2d:d8:0e:de /usr/local/lib/opensc-pkcs11.so (RSA)
```

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. How to Configure the Client to Use 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 the NoMachine client on your device:
-Create a new session or select an existing one. Click on its icon to open the context menu and click on 'Edit connection'.
-Choose Protocol SSH and click on the Advanced button.
-Select 'Smart card' and open its Settings. -The client 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'.
-Confirm your settings and connect. Enter your PIN when you will be prompted for.

The session should now be up and running.

TIP

Select the 'Forward authentication' checkbox in the Smart card's settings panel if you want to forward credentials into the session.

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:

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

These three methods will be looked at in detail below.

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 the following command and verify if 'F' is present among the flags:

```
$ klist -f
```

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/
```

5.5. How to Set-up PAM and Kerberos on Server Host

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

```
$ 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 use 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

#Pkinit 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 (replace <REALM_NAME> and with the proper realm and principal.:

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

TIP

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

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

If you want users to authenticate with key obtained by pkinit (e.g. smartcard) and not with password, modify principals to use pre_auth:

```
$ 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_identitY' 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. How to Configure the Client to Use Kerberos Ticket

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

To be sure you have a valid Keberos ticket, first of all run the following command, output should be similar to this:

```
$ klist

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. Click on its icon to open the context menu and click on 'Edit connection'.
-Choose Protocol SSH or NX and click on the Advanced button.
-Select 'Kerberos' and open its Settings. -Choose any of the Kerberos options as authentication method.
-Confirm your settings and connect. You should be prompted for your username.

TIP

Select the 'Forward authentication' checkbox in the Kerberos settings panel if you want to forward credentials into the session.

5.8. How to Forward Kerberos Ticket to Remote

This paragraph applies to a multi-node environment made of NoMachine Enterprise Terminal Server and Terminal Server Node(s).

Pre-requisite
You need to have installed and configured Kerberos properly on server host. You need also 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:

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

Check if you have valid and forwardable Keberos ticket. The output of this command should be similar to:

```
$ klist -f

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. How to Run a Session with Kerberos Ticket and PAM Module on the 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:

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

Run NoMachine client on your device:

-Create a new session or select an existing one. Click on its icon to open the context menu and click on 'Edit connection'.
-When 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 the following command. Output should be similar to:

```
$ klist

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
```

6. Use SSH Kerberos Authentication on Windows (Client Side)

6.1. How To Install Kerberos

If you don't use AD (Active Domain), you need to install and configure Kerberos MIT on the client host.

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

TIP

If you have AD, you can either use MIT Kerberos as explained here, or use Microsoft SSPI. The NoMachine client automatically switches to Microsoft SSPI if MIT Kerberos is not detected.

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"

NOMACHINE.COM is just an example, change it to your local domain/realm.

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:
- Run the **netidmgr.exe** tool with administrative privileges.
- 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.
