SafeNet ProtectDB for Teradata

User Guide

gemalto
security to be free

Product Version: 8.5.0

Document Number: 007-013889-001, Rev A

Release Date: September 15, 2017

Table of Contents

About This Guide

This introductory chapter gives a brief summary of the book’s contents, identifies the audience, explains how to best use the written material, and discusses the documentation conventions used.

This chapter contains the following sections:

What is in This Guide?

Generally speaking, SafeNet user guides are written for network administrators, security engineers, database administrators, application developers, and other technology professionals responsible for daily operations in support of data security. The written material we provide describes how to configure and operate SafeNet’s products and assumes a working knowledge of networking, computer security, database management, and cryptography.

This specific book is designed for advanced database administrators familiar with Teradata databases, the SafeNet KeySecure appliance, and the general system architecture of the SafeNet KeySecure appliance as documented in the *SafeNet KeySecure Appliance User Guide*.

Organization

This guide is arranged as follows:

Chapter 1, **“Overview”**

Presents an overview of how SafeNet ProtectDB integrates with current application. Topics discussed include using SafeNet ProtectDB, supported platforms and data types.

Chapter 2, **“Installing SafeNet ProtectDB on SUSE Linux”**

Describes the instructions to install and set up SafeNet ProtectDB.

Chapter 3, **“Configuring the Properties File”**

Details the process of modifying the properties file.

Chapter 4, **“Creating User Mappings”**

Describes the user mapping and provides information on its creation/modification/removal.

Chapter 5, **“Setting up SSL”**

Describes how to set up the SafeNet KeySecure and SafeNet ProtectDB to use SSL.

Chapter 6, **"Standard Encryption and Decryption"**

Describes the user defined functions provided with SafeNet ProtectDB for Teradata for standard encryption/decryption.

Chapter 7, **"FPE Encryption and Decryption"**

Describes the user defined functions provided with SafeNet ProtectDB for Teradata for format preserving encryption/decryption.

Conventions

The following paragraph formats are used to highlight information in the text:

NOTE: A note conveys information that supplements the preceding text. This information may refer to certain situations or a specific technical setup.

IMPORTANT ! An important note is a very significant piece of information required for the completion of a task.

TIP ! A tip helps apply the information in the preceding text.

WARNING ! A warning advises to exercise care when working around specified equipment conditions. Heeding a warning can prevent personal injury, system disruption, or equipment damage.

1 Overview

SafeNet ProtectDB for Teradata allows you to protect sensitive data in Teradata database tables by managing a seamless interaction between the database and the SafeNet KeySecure. SafeNet ProtectDB for Teradata supports both standard and format preserving encryption/decryption.

This chapter covers the following topics:

NOTE: This user guide at times may refer SafeNet ProtectDB for Teradata as ProtectDB.

Using SafeNet ProtectDB

SafeNet ProtectDB enables your database to communicate directly with a KeySecure appliance; and encrypt and decrypt data. The database uses the ProtectDB to authenticate users and submit cryptographic requests to the KeySecure. The KeySecure then performs the cryptographic request and returns the resulting data.

Integrated with this process are various levels of security that protect your data from unauthorized users. To access the cryptographic features needed to view your encrypted data, applications and users must have permission to access the crypto key on the SafeNet KeySecure.

Supported Platforms

The SafeNet KeySecure for Teradata supports Teradata database 14.0 running on SUSE Linux.

Supported Data Types

SafeNet ProtectDB for Teradata supports the following data types for standard encryption and format preserving encryption (FPE):

Standard Encryption	Format Preserving Encryption
CHAR	CHAR
VARCHAR	VARCHAR
INT	INT
BYTEINT	BYTEINT
SMALLINT	SMALLINT
DATE	
TIME	
TIMESTAMP	

NOTE: For FPE operation, it is recommended to use minimum data length of two bytes. For more information on FPE operations, refer Chapter 7, “FPE Encryption and Decryption”.

2 Installing SafeNet ProtectDB on SUSE Linux

This chapter contains detailed instructions on how to install and set up SafeNet ProtectDB for Teradata on SUSE Linux. This chapter contains the following information:

Preparing for the Installation

Before running the installation script:

1. Create the `/usr/local/ingrian` directory on all the nodes.
2. Change the ownership of `/usr/local/ingrian` on all the nodes so that it is owned by the user and group that will maintain the CLI connector (for example, user is `tdatuser` and group is `tdatudf`).
3. Create the `INGRIAN` user in the database.
4. Grant `create function` permission to the `INGRIAN` user.
5. Confirm that the database is running and that you can access it via BTEQ.

Installing SafeNet ProtectDB

To install SafeNet ProtectDB:

1. Download and unzip the SafeNet ProtectDB software on a Teradata node. This will extract the following files to `/SafeNet/ProtectDB-Teradata`:

File Name	Description
BOM	The bill of materials file. Lists the files installed when install.sh is run.
IngrianNAE.properties	This file contains the parameters used to communicate with the KeySecure. This file will be moved when you run `install.sh`.
ingtdcli	The command line tool that enables you to create, delete, and list user mappings; and list encrypted tables and columns in the database.
install.sh	This script installs the SafeNet ProtectDB software.

File Name	Description
libIngICAPI.so-8.4.2.000	This library contains the functions used to communicate with the KeySecure.
libingtd.so	This library contains the UDFs that allow the encryption and decryption of strings.
passwd	This file contains the passwords and user names used by your user mappings. The file is encoded; the information is not in cleartext.
test.btq	This file contains sql code to test UDFs once they are installed. You must first run testtable.btq.
testtable.btq	This file contains sql code to create and populate a sample table in the database.
uninstall.sh	This script removes the SafeNet ProtectDB software.
xsp.btq	This file contains the sql used to create the UDFs in the database during the installation.
xspdrop.btq	This file contains the sql used to remove the UDFs from the database during the uninstallation.

2. Run install.sh. The script does the following:
   - Places the following files in `/usr/local/ingrian`:
     - IngrianNAE.properties
     - ingtdcli
     - libIngICAPI.so-8.4.2.000
     - passwd
   - Creates the instance ID and stores it in `/usr/local/ingrian/instanceid`.
   - Creates the following UDFs in the database:

FileName	Description
`SFNT_ENCRYPT_CHAR`	Encrypts data (of the corresponding data types) using the key name, fully qualified algorithm, and IV passed to the function. These UDFs are used for standard encryption.
`SFNT_ENCRYPT_INT`	
`SFNT_ENCRYPT_SMLINT`	
`SFNT_ENCRYPT_BINT`	
`SFNT_ENCRYPT_DAT`	
`SFNT_ENCRYPT_TIME`	
`SFNT_ENCRYPT_TIMESTMP`	
`SFNT_DECRYPT_CHAR`	Decrypts data (of the corresponding data types) using the key name, fully qualified algorithm, and IV passed to the function. These UDFs are used for standard decryption.
`SFNT_DECRYPT_INT`	
`SFNT_DECRYPT_SMLINT`	
`SFNT_DECRYPT_BINT`	
`SFNT_DECRYPT_DAT`	
`SFNT_DECRYPT_TIME`	
`SFNT_DECRYPT_TIMESTMP`	

FileName	Description
`SFNT_ENCRYPT_CHARFPE`	Encrypts data (of the corresponding data types) using the key name, fully qualified algorithm, IV, tweak algorithm and tweak data passed to the function. These UDFs are used for FPE encryption.
`SFNT_ENCRYPT_INTFPE`	
`SFNT_ENCRYPT_ALPHANUMFPE`	
`SFNT_DECRYPT_CHARFPE`	Decrypts data (of the corresponding data types) using the key name, fully qualified algorithm, IV, tweak algorithm and tweak data passed to the function. These UDFs are used for FPE decryption.
`SFNT_DECRYPT_INTFPE`	
`SFNT_DECRYPT_ALPHANUMFPE`	

- Creates the `sfnt` folder in the `/var/log` directory for logging.

3. Modify the `IngrianNAE.properties` file to suit your environment.

   For example, to use the `ALPHANUMFPE` UDFs, set the `Symmetric_Key_Cache_Enabled = tcp_ok` in the `IngrianNAE.properties` file.

4. Create a user on the KeySecure. (Refer to the *SafeNet KeySecure Appliance User Guide* to create a KeySecure user.)

5. Create a user mapping to associate a database user (user name should be in `CAPS`) with a KeySecure user (case sensitive). The following syntax is used to create a user mapping:

   ```
   (Linux)$ ./ingtdcli mapuser dbuser=<Teradata User> naeuser=<KeySecure User>
   naepassword=<KeySecure User Password>
   ```

   For more information, refer to Chapter 4, "Creating User Mappings" on page 20 about the creation, modification and removal of user mapping.

6. Change the ownership of `/var/log/sfnt` directory and all the files in the `/usr/local/ingrian` directory so that it is owned by the user and group that will maintain the CLI connector (for example, user is `tdatuser` and group is `tdatudf`). The following command is an example of changing the ownership of all the files in the `/usr/local/ingrian` directory to `tdatuser` and `tdatudf`:

   ```
   chown tdatuser:tdatudf *
   ```

   NOTE: If SSL connection between SafeNet KeySecure and ProtectDB is to be set up, refer to Chapter 5, "Setting up SSL" on page 22.

7. Install ProtectDB on other Teradata nodes:

   a. Create `/usr/local/ingrian` directories on all the nodes, if not created.

   a. Copy the contents of the `/usr/local/ingrian` directory from the primary node (the first installation) to other Teradata nodes using the following command:

      ```
      pcl -send file_name package_directory
      ```

   b. Create the `/var/log/sfnt` directory for logging in all the nodes using the following command:

      ```
      pcl -shell mkdir /var/log/sfnt
      ```

   You may also need to set correct ownership and permissions after copying the files using the following commands:

   `pcl -shell chown owner:group full_path_of_package_file` (for example, owner is `tdatuser` and group is `tdatudf`)

```
pcl -shell chmod permissions full_path_of_package_file
```

`pcl -shell chown tdatuser:tdatudf /var/log/sfnt` (to change the ownership of `sfnt` directory to `tdatuser`)

NOTE: Whenever the user mapping is modified on any node, the `passwd` file needs to be copied to all the nodes again. Database restart is not required in this case.

NOTE: After any modification on the files in the `/usr/local/ingrian` directory, it should be replaced in all the nodes. Additionally, if the `IngrianNAE.properties` file is modified, the database service needs to be restarted along with replacing it in all the nodes.

8. Restart the database by running the following commands:

```
/etc/init.d/tpa stop
/etc/init.d/tpa start
```

Upgrading SafeNet ProtectDB

The SafeNet ProtectDB software is regularly updated to include new features and provide fixes to any issues occurring in previous releases. Upgrading the existing SafeNet ProtectDB software brings the new features added and enhancements made to the product since the SafeNet ProtectDB installation.

To upgrade to the latest release of SafeNet ProtectDB, uninstall the installed ProtectDB software and install the latest SafeNet ProtectDB software.

Uninstalling SafeNet ProtectDB

To uninstall the ProtectDB software, run the `uninstall.sh` file on the node (where the first installation was done).

NOTE: Decrypt any data before uninstsalling the ProtectDB. Uninstalling the ProtectDB deletes all the files in the `/usr/local/ingrian` directory.

Remove the `/usr/local/ingrian` and `/var/log/sfnt` directories from all other nodes.

To remove the `ingrian` and `sfnt` directories from other nodes, run the following commands:

```
pcl -shell rm -rf /var/log/sfnt
pcl -shell rm -rf /usr/local/ingrian
```

3
Configuring the Properties File

This chapter lists the contents of the IngrianNAE.properties file. The properties file defines, among other things, the IP address, port, and protocol of the SafeNet KeySecure to which your client connects. This chapter contains the following topics:

Editing the Properties File

The values in the properties file are case-sensitive. yes is not YES, or Yes. tcp is not TCP or Tcp. Follow the example of the default properties file.

You can comment-out values using #. You will see that the properties file is delivered with **Cipher_Spec** commented-out. You may want to use comments to save settings when troubleshooting. For example, you could store commonly used NAE_IP addresses like this:

```
NAE_IP=10.0.0.2
#NAE_IP=10.0.0.3
#NAE_IP=10.0.0.4
```

When editing parameters that use time values, you can use the following abbreviations:

- ms - Milliseconds. e.g. 4500ms for 4.5 seconds.
- s - Seconds. e.g. 30s for 30 seconds.
- m - Minutes. e.g. 5m for 5 minutes.
- h - Hours. e.g. 10h for 10 hours.
- d - Days. e.g. 2d for 2 days.

If you do not include an abbreviation, the default time unit is used. For most time-related values, the default is *milliseconds*. For **Symmetric_Key_Cache_Expiry** and **Persistent_Cache_Expiry_Keys**, the default is seconds.

NOTE: If the `IngrianNAE.properties` file is modified after ProtectDB installation, the database service needs to be restarted along with replacing it in all the nodes.

Parameters

Once you install the client software, you can customize it to meet the needs of your environment by modifying the properties file. To customize the software, you need to modify the properties file.

The parameters listed in the file, including the delivered settings, are listed below.

```
Version=3.1
NAE_IP=
NAE_Port=9000
Protocol=tcp
Use_Persistent_Connections=yes
Size_of_Connection_Pool=300
Connection_Timeout=60000
Connection_Idle_Timeout=600000
Connection_Retry_Interval=600000
Cluster_Synchronization_Delay=100
CA_File=
Cert_File=
Key_File=
Passphrase=
Symmetric_Key_Cache_Enabled=no
Symmetric_Key_Cache_Expiry=43200
Log_Level=
Log_File=
```

NOTE: Some other parameters are listed in the `IngrianNAE.properties` file that are not supported and not mentioned in the preceding list.

Version

The `Version` parameter indicates the version of the properties file and should not be modified.

NAE_IP

The `NAE_IP` parameter specifies the IP address of the SafeNet KeySecure.

NAE_Port

The `NAE_Port` specifies the port of the SafeNet KeySecure. The default port is `9000`.

IMPORTANT ! Clients and servers must use the same port.

Protocol

The `Protocol` parameter specifies the protocol used to communicate between the client and the SafeNet KeySecure.

Possible settings:

- **tcp**

- **ssl** – The ssl option uses TLSv1.2. To establish SSL connections between your NAE clients and servers TLSv1.2 should be enabled on your servers.

  IMPORTANT ! Clients and servers must use the same protocol. If your SafeNet KeySecure servers are listening to SSL requests, and your clients are not sending SSL requests, you will run into problems.

TIP ! *SafeNet recommends that you gradually increase security after confirming connectivity* between the client and the SafeNet KeySecure. Once you have established a TCP connection between the client and the server, it is safe to move on to SSL. Initially configuring a client under the most stringent security constraints can complicate troubleshooting.

Use_Persistent_Connections

The `Use_Persistent_Connections` parameter enables the persistent connections functionality.

Possible settings:

- **yes** – Enables the feature. The client establishes persistent connections with the NAE Servers. This is the default value.

- **no** – Disables the feature. A new connection is made for each connection request. The connection is closed as soon as the client receives the server response.

Size_of_Connection_Pool

The `Size_of_Connection_Pool` parameter specifies the total number of client-server connections that your configuration could possibly allow. (Not what actually exists at a given moment.)

Possible settings:

- **Any positive integer** – The default is `300`.

Connections in the pool can be active or waiting, TCP or SSL. A connection is created as needed, and the pool scales as needed. The pool starts at size 0, and can grow to the value set here. Once the pool is full, new connection requests must wait for an existing connection to close.

Connection pooling is configured on a per-client basis. The size of the pool applies to each *client*; it is not a total value for a SafeNet KeySecure or for a load balancing group. If there are multiple clients running on the same machine, separate connection pools are maintained for each client.

Connection_Timeout

The `Connection_Timeout` parameter specifies how long the client waits for the TCP connect function before timing out.

Possible settings:

- **0** – Disables this setting. The client uses the operating system's connect timeout.

- **Any positive integer** – The default is `60000ms`.

Setting this parameter a few hundred ms *less* than the operating system's connection timeout makes connection attempts to a downed server fail faster, and failover happens sooner. If a connection cannot be made before the timeout expires, the server is marked as down and taken out of the rotation.

NOTE: If your client is working with many versions of a key, do not set the **Connection_Timeout** parameter too low. Otherwise, the client connection may close before the operation is complete.

Connection_Idle_Timeout

The `Connection_Idle_Timeout` parameter specifies the amount of time connections in the connection pool can remain idle before the client closes them.

Possible settings:

- **Any positive integer** – The default is `600000ms` (10 min).

  IMPORTANT ! There are two different connection timeout values: *one on the SafeNet KeySecure*, and *one in the properties file*. The value of the timeout in the properties file must be less than what is set on the server. This lets the client control when idle connections are closed. Otherwise, the client can maintain a connection that is closed on the server side, which can lead to error.

Connection_Retry_Interval

The `Connection_Retry_Interval` parameter determines how long the client waits before trying to reconnect to a disabled server. If one of the SafeNet KeySecure servers in a load balanced configuration is not reachable, the client assumes that the server is down, and then waits for the specified time period before trying to connect to it again.

Possible settings:

- **0** – This is the infinite retry interval. The disabled server will never be brought back into use.
- **Any positive integer** – The default value is `600000ms` (10 minutes).

Cluster_Synchronization_Delay

The `Cluster_Synchronization_Delay` parameter specifies how long the client will wait before assuming that key changes have been synchronized throughout a cluster. After creating, cloning, importing or modifying a key, the client will continue to use the same SafeNet KeySecure appliance until the end of this delay period.

Possible settings:

- **0** – Disables the function.
- **Any positive integer** – The default is `100s`. You may want a higher setting for large clusters.

For example, the client sets `Cluster_Synchronization_Delay` to 100s and sends a key creation request to appliance A, which is part of a cluster. Appliance A creates the key and automatically synchronizes with rest of the cluster. The client will use only appliance A for 100 seconds - enough time for the cluster synchronization to complete. After this time period, the client will use other cluster members as before.

Cluster_Synchronization_Delay

The `Cluster_Synchronization_Delay` parameter specifies how long the client will wait before assuming that key changes have been synchronized throughout a cluster. After creating, cloning, importing, or modifying a key, the client will continue to use the same KeySecure appliance until the end of this delay period.

Possible settings:

- **0** – Disables the function.
- **Any positive integer** – The default is `100s`. You may want a higher setting for large clusters.

For example, the client sets `Cluster_Synchronization_Delay` to 100s and sends a key creation request to appliance A, which is part of a cluster. Appliance A creates the key and automatically synchronizes with rest of the cluster. The client will use only appliance A for 100 seconds - enough time for the cluster synchronization to complete. After this time period, the client will use other cluster members as before.

CA_File

The `CA_File` parameter refers to the CA certificate that was used to sign the server certificate presented by the NAE Server to the client.

Possible settings:

- **The path and file name** – The path can be absolute or relative to your application. Do not use quotes, even if the path contains spaces.

Because all SafeNet KeySecure servers in a clustered environment must have an identical configuration, all servers in the cluster use the same server certificate. As such, you need to point to only one CA certificate in the `CA_File` system parameter. If you do not supply the CA certificate that was used to sign the server certificate used by the SafeNet KeySecure servers, your client applications cannot establish SSL connections with any of the servers in the cluster.

File paths can be absolute or relative. Unless otherwise noted, when prompted for a file, you should specify both a path and file name.

If a local CA on the SafeNet KeySecure was used to sign the NAE Server certificate, you can download the certificate for the local CA, and put that certificate on the client.

Cert_File

The `Cert_File` parameter stores the path and file name of the client certificate. This is used only when your SSL configuration requires clients to provide a client certificate to authenticate to the SafeNet KeySecure servers.

Possible settings:

- **The path and file name** – The path can be absolute or relative to your application. Don’t use quotes, even if the path contains spaces. Client certificates *must* be PEM encoded.

  IMPORTANT ! If this value is set, the certificate and private key must be present, even if the SafeNet KeySecure is not configured to request a client certificate.

Key_File

The `Key_File` parameter refers to the private key associated with the client certificate specified in the `Cert_File` parameter.

Possible settings:

- **The path and file name** – The path can be absolute or relative to your application. Do not use quotes, even if the path contains spaces. The client private key must be in PEM-encoded PKCS#12 format.

Because this key is encrypted, you must use the **Passphrase** parameter so the SafeNet KeySecure can decrypt it.

IMPORTANT ! If this value is set, the certificate and private key must be present, even if the SafeNet KeySecure is not configured to request a client certificate.

Passphrase

The `Passphrase` parameter refers to the passphrase associated with the private key.

Possible settings:

- **The passphrase associated with the private key named in `Key_File`.**

If you do NOT provide this passphrase, the client attempts to read the passphrase from standard input; this causes the application to hang.

IMPORTANT ! Remember that the properties file is NOT encrypted. Make sure that this file resides in a secure directory and has appropriate permissions so that it is readable only by the appropriate application or user.

Symmetric_Key_Cache_Enabled

The `Symmetric_Key_Cache_Enabled` parameter determines if the symmetric key caching feature is enabled. *Only symmetric keys can be cached.*

Possible settings:

- **no** – Key caching is disabled. Remote encryption (encryption performed on SafeNet KeySecure) is available as normal.
- **yes** – Key caching is enabled and the NAE XML protocol is used for exporting keys. The `Protocol` parameter must be set to ssl.
- **tcp_ok** – Key caching is enabled over both tcp and ssl connections. The NAE XML protocol is used for exporting keys.

NOTE: When `yes` or `tcp_ok` option is used for local mode operations, the key's group permission policy is not applicable.

Symmetric_Key_Cache_Expiry

The `Symmetric_Key_Cache_Expiry` parameter determines the *minimum* amount of time that a key remains in the client key cache.

Possible settings:

- **0** – This is the infinite timeout setting. Keys are never purged from the client cache.
- **A positive integer** – At the end of this interval, the key is purged from the cache the next time the library is called. The default value is `43200` **seconds** (12 hours).

Log_Level

The `Log_Level` parameter determines the level of logging performed by the client.

NOTE: The `Log_Level` parameter is not supported in this version of ProtectDB for Teradata.

Log_File

The `Log_File` parameter specifies a name (and possibly a path) for the log file. This parameter is not supported in the current release.

NOTE: The log file `SFNT.TRACE` is generated in the `/var/log/sfnt` directory. The `tdatuser` should be the owner of `/var/log/sfnt` directory for logging.

4 Creating User Mappings

This chapter explains how to use the ingtdcli tool to create user mappings. This chapter contains the following information:

Overview

A user mapping associates a database user with a KeySecure user. You need a user mapping to encrypt and decrypt data. The database user must be able to access the data you are manipulating. The KeySecure user must be able to access the key you want to use.

The ingtdcli tool enables you to create, view, modify, and remove user mappings.

Creating a User Mapping

To create a user mapping, run the ingtdcli configuration tool. Linux users will find the tool in `/usr/local/ingrian`. Use the following syntax:

```
(Linux)$ ./ingtdcli mapuser dbuser=<Teradata User> naeuser=<KeySecure User>
naepassword=<KeySecure User Password>
```

For example,

```
(Linux)$ ./ingtdcli mapuser dbuser=<TERA1> naeuser=<ks1> naepassword=<asdf1234>
```

The naeuser and naepassword are encrypted and stored in the passwd file.

NOTE: Ensure that the dbuser name used is in `CAPS`.

To perform encryption/decryption, `user mapping` should be correct and CLI user should have read permission on the `passwd` file, otherwise, the following error is thrown:

```
Unable to fetch Authorization information for user.
```

NOTE: If a new user mapping is created after ProtectDB installation, the `passwd` file needs to be copied to all the nodes again. Database restart is not required in this case.

Viewing Existing User Mappings

To see the list of current user mappings, use the ingtdcli tool with the `listusers` command:

```
(Linux)$ ./ingtdcli listusers
```

Modifying a User Mapping

To modify an existing user mapping, use the ingtdcli tool with the `mapuser` command:

```
(Linux)$ ./ingtdcli mapuser dbuser=<Teradata User> naeuser=<KeySecure User>
naepassword=<KeySecure User Password>
```

For example,

```
(Linux)$ ./ingtdcli mapuser dbuser=<TERA1> naeuser=<ks5> naepassword= <qwerty>
```

NOTE: If a user mapping is modified after ProtectDB installation, the `passwd` file needs to be copied to all the nodes again. Database restart is not required in this case.

Removing a User Mapping

To remove a user mapping, use the ingtdcli tool with the `unmapuser` command:

```
(Linux)$ ./ingtdcli unmapuser dbuser=<Teradata User>
```

For example,

```
(Linux)$ ./ingtdcli unmapuser dbuser=<TERA1>
```

NOTE: If a user mapping is removed after ProtectDB installation, the `passwd` file needs to be copied to all the nodes again. Database restart is not required in this case.

5 Setting up SSL

This chapter provides an overview of the SSL and SSL with Client Certificate Authentication features, and provides a walkthrough of both configuration procedures. This chapter contains the following topics:

SSL Overview

Standard SSL communication requires a certificate that identifies the server. This certificate is signed by a certificate authority (CA) known to both the server and the client. During the SSL handshake, the server certificate is passed to the client. The client uses a copy of the CA certificate to validate the server certificate, thus authenticating the server.

While the CA can be a third-party CA or your corporate CA, you will most likely use a local CA on the SafeNet KeySecure appliance. If you are not using a local CA, consult your CA documentation for instructions on signing requests and exporting certificates.

TIP ! SafeNet recommends that you increase security *only after* confirming network connectivity. You should establish a TCP connection before enabling SSL. Otherwise, an unrelated network connection mistake could interfere with your SSL setup and complicate the troubleshooting process.

To use an SSL connection when communicating with the SafeNet KeySecure appliance, you must configure both the server and the client.

To configure the server, you must:

1. Create a server certificate. (If you're using a cluster, each member must have its own, unique certificate.)

   This may involve the following steps:

   a. Creating a Local CA.

b. Creating a Server Certificate Request on the SafeNet KeySecure Management Console.

c. Signing a Server Certificate Request with a Local CA.

2. Update the Cryptographic Key Server settings on the SafeNet KeySecure Management Console (Device, Key Server, Key Server).

   You'll need to check **Use SSL** and select your server certificate in the **Server Certificate** field.

To configure the client, you must:

1. Place a copy of the CA certificate on your client on all the Teradata nodes.

   This may involve the following step:

   - Downloading the Local CA Certificate.

2. Update `IngrianNAE.properties` file as follows:

   - `Protocol=ssl`
   - `CA_File=<location and name of the CA certificate file>`

   NOTE: Keep the server certificate file in same directory on all the nodes. Recommended location is `/usr/local/ingrian` directory on all the nodes. Also, `tdatuser` and `tdatudf` should have ownership of the server certificate file and the directory containing the certificate.

   NOTE: Whenever you update the properties file, you must copy the `IngrianNAE.properties` file to all the nodes and must restart the database for the changes to take effect.

SSL Configuration Procedures

This section describes the procedures you will follow when configuring SSL. It explains the following processes:

- Creating a Local CA
- Creating a Server Certificate Request on the SafeNet KeySecure Management Console
- Signing a Server Certificate Request with a Local CA
- Importing a Server Certificate to the SafeNet KeySecure Appliance
- Downloading the Local CA Certificate

Creating a Local CA

To create a local CA:

1. Log on to the SafeNet KeySecure Management Console as an administrator with Certificate Authorities access control.
2. Navigate to the Create Local Certificate Authority section on the Certificate and CA Configuration page (Security, CAs & SSL Certificates, Local CAs).
3. Modify the fields as needed.

4. Select either ***Self-signed Root CA*** or ***Intermediate CA Request*** as the **Certificate Authority Type**.
5. Click **Create**.

NOTE: Only a local CA can sign certificate requests on the SafeNet KeySecure appliance. If you are using a CA that does not reside on the SafeNet KeySecure appliance, you cannot use the SafeNet KeySecure Management Console to sign certificate requests.

Creating a Server Certificate Request on the SafeNet KeySecure Management Console

To create a server certificate request on the SafeNet KeySecure Management Console:

1. Log on to the SafeNet KeySecure Management Console as an administrator with Certificates access control.
2. Navigate to the Create Certificate Request section of the Certificate Configuration page (Security, CAs & SSL Certificates, SSL Certificates) and modify the fields as needed.
3. Click **Create Certificate Request**. This creates the certificate request and places it in the Certificate List section of the Certificate and CA Configuration page. The new entry shows that the **Certificate Purpose** is *Certificate Request* and that the **Certificate Status** is *Request Pending*.

Signing a Server Certificate Request with a Local CA

To sign a server certificate request with a local CA:

1. Log on to the SafeNet KeySecure Management Console as an administrator with Certificates and Certificate Authorities access controls.
2. Navigate to the Certificate List section on the Certificate and CA Configuration page (Security, CAs & SSL Certificates, SSL Certificates).
3. Select the certificate request and click **Properties**.
4. Copy the text of the certificate request. The copied text must include the header (-----BEGIN CERTIFICATE REQUEST-----) and footer (-----END CERTIFICATE REQUEST-----).
5. Navigate to the Local Certificate Authority List (Security, CAs & SSL Certificates, Local CAs). Select the local CA and click **Sign Request** to access the Sign Certificate Request section.
6. Modify the fields as shown:
   - **Sign with Certificate Authority** - Select the CA that signs the request.
   - **Certificate Purpose** - Select **Server**.
   - **Certificate Duration (days)** - Enter the life span of the certificate.
   - **Certificate Request** - Paste all text from the certificate request, including the header and footer.
7. Click **Sign Request**. This will take you to the CA Certificate Information section.
8. Copy the actual certificate. The copied text must include the header (-----BEGIN CERTIFICATE-----) and footer (-----END CERTIFICATE-----).
9. Navigate back to the Certificate List section (Security, CAs & SSL Certificates, SSL Certificates). Select your certificate request and click **Properties**.
10. Click **Install Certificate**.

11. Paste the actual certificate in the Certificate Response text box. Click **Save**. The SafeNet KeySecure Management Console returns you to the Certificate List section. The section will now show that the **Certificate Purpose** is *Server* and that the **Certificate Status** is *Active*.

The certificate can now be used as the server certificate for the NAE Server.

Importing a Server Certificate to the SafeNet KeySecure Appliance

As an alternative to the certificate creation procedure outlined above, you can import a certificate to the SafeNet KeySecure appliance.

To import a certificate to the SafeNet KeySecure appliance:

1. Log on to the SafeNet KeySecure Management Console as an administrator with Certificates access control.
2. Navigate to the Import Certificate section of the Certificate and CA Configuration page (Security, CAs & SSL Certificates, SSL Certificates).
3. Select the method used to import the certificate file.
4. Enter the name of the file and the private key password.
5. Click the **Import Certificate** button.

The certificate can now be used as the server certificate for the NAE Server.

Downloading the Local CA Certificate

To download a local CA certificate from the SafeNet KeySecure appliance:

1. Log on to the SafeNet KeySecure Management Console as an administrator with Certificate Authorities access control.
2. Navigate to the Local Certificate Authority List section of the Certificate and CA Configuration page (Security, CAs & SSL Certificates, Local CAs).
3. Select the Local CA and click the **Download** button to download the file to your client. You should place the CA certificate in a secure location and modify access appropriately.

NOTE: Use the `CA_File` parameter in the `IngrianNAE.properties` file to indicate the name and location of the CA certificate. You must copy the `IngrianNAE.properties` file to all the nodes after any modification and must restart the database for the changes to take effect.

SSL Walkthrough for SafeNet Clients

This walkthrough assumes the following:

- You have read the SSL overview section.
- You have configured a TCP connection between your client and the SafeNet KeySecure appliance.

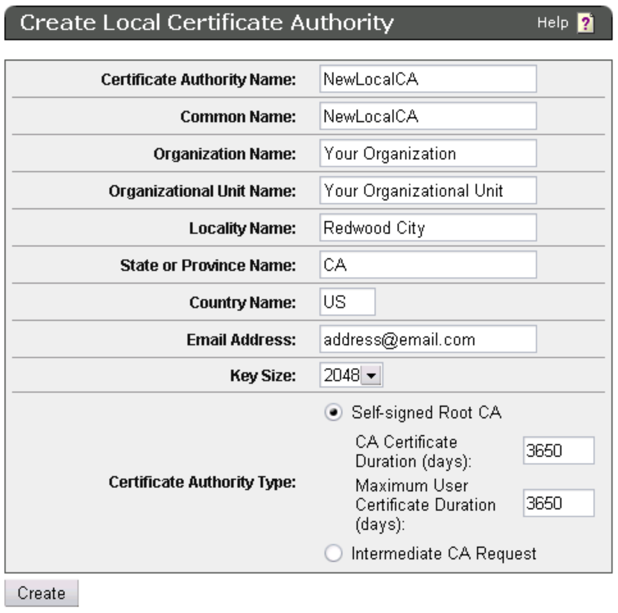
There are a few different ways that you could configure SSL. For example, you can use a CA that does not reside on the SafeNet KeySecure appliance or you can create a new CA. This walkthrough makes such decisions for you. By following these instructions, you will:

- Create a Local CA.
- Create a Certificate Request.
- Create a Server Certificate by signing the Certificate Request with the Local CA.
- Download the Local CA to the client.

Once you have completed and understood this walkthrough, you might decide to alter some steps to better fit your organization's policies.

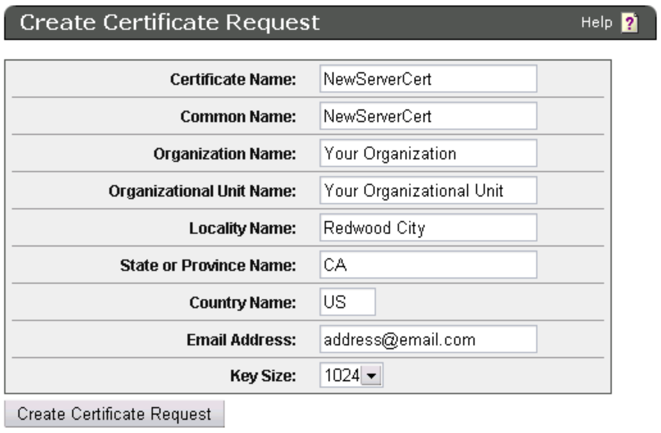
To configure SSL:

1. Log on to the SafeNet KeySecure Management Console as an administrator with Certificates, Certificate Authorities, and NAE Server access controls.
2. Navigate to the Create Local Certificate Authority section (Security, CAs & SSL Certificates, Local CAs). Enter the values shown below to create a new local CA. Click **Create**.



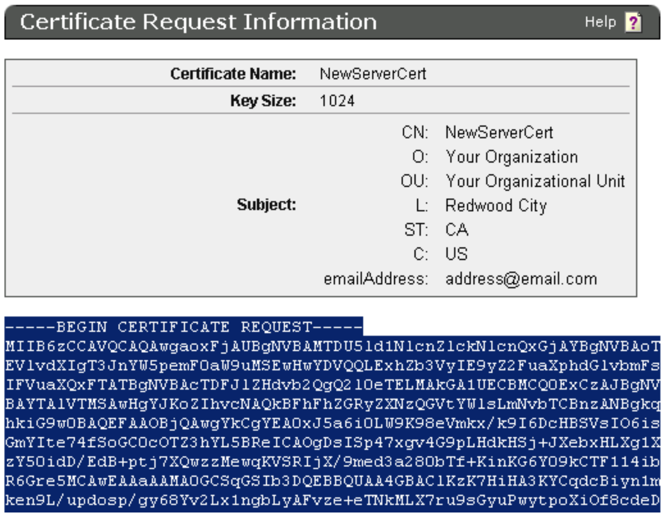
Certificate Authority Name:	NewLocalCA
Common Name:	NewLocalCA
Organization Name:	Your Organization
Organizational Unit Name:	Your Organizational Unit
Locality Name:	Redwood City
State or Province Name:	CA
Country Name:	US
Email Address:	address@email.com
Key Size:	2048
Certificate Authority Type:	[✓ Self-signed Root CA] CA Certificate Duration (days): 3650; Maximum User Certificate Duration (days): 3650; Intermediate CA Request

3. Navigate to the Create Certificate Request section (Security, CAs & SSL Certificates, SSL Certificates). Enter the values shown below to create a request. Click **Create Certificate Request**.



Certificate Name:	NewServerCert
Common Name:	NewServerCert
Organization Name:	Your Organization
Organizational Unit Name:	Your Organizational Unit
Locality Name:	Redwood City
State or Province Name:	CA
Country Name:	US
Email Address:	address@email.com
Key Size:	1024

4. Select your new certificate request from the Certificate List section (right above the Create Certificate Request section). Click **Properties**. Copy the actual request (highlighted below). Include the header and footer.

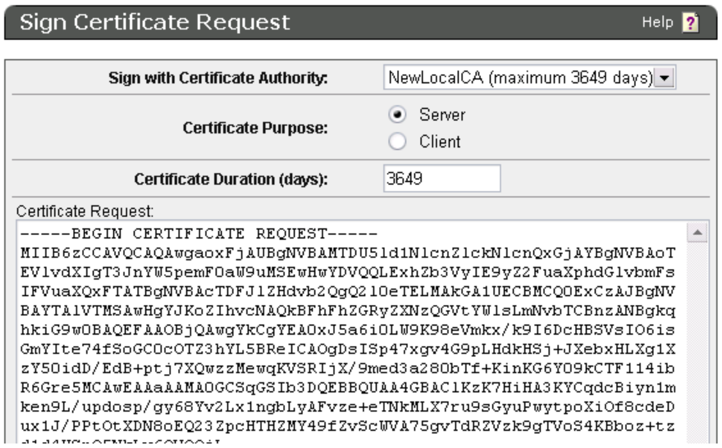


Certificate Name:	NewServerCert
Key Size:	1024
Subject:	CN: NewServerCert O: Your Organization OU: Your Organizational Unit L: Redwood City ST: CA C: US emailAddress: address@email.com

```
-----BEGIN CERTIFICATE REQUEST-----
MIIB6zCCAVQCAQAwgaoxFjAUBgNVBAMTDU5ld1NlcnZlckNlcnQxGjAYBgNVBAoT
EVlvdXIgT3JnYW5pemF0aW9uMSEwHwYDVQQLExhZb3VyIE9yZ2FuaXphdGlvbmFs
IFVuaXQxFTATBgNVBAcTDFJlZHdvb2QgQ2l0eTELMAkGA1UECBMCQ0ExCzAJBgNV
BAYTAlVTMSAwHgYJKoZIhvcNAQkBFhFhZGRyZXNzQGVtYWlsLmNvbTCBnzANBgkq
hkiG9w0BAQEFAAOBjQAwgYkCgYEA0xJ5a6iOLW9K98eVmkx/k9I6DcHBSVsIO6is
GmYIte74fSoGCOcOTZ3hYL5BReICAOgDsISp47xgv4G9pLHdkHSj+JXebxHLXg1X
zY5OidD/EdB+ptj7XQwzzMewqKVSRIjX/9med3a28ObTf+KinKG6YO9kCTF114ib
R6Gre5MCAwEAAaAAMA0GCSqGSIb3DQEBBQUAA4GBAClKzK7HiHA3KYCqdcBiyn1m
ken9L/updosp/gy68Yv2Lx1ngbLyAFvze+eTNkMLX7ru9sGyuPwytpoXiOf8cdeD
```

5. Navigate back to the Local Certificate Authority List section (Security, CAs & SSL Certificates, Local CAs). Select your new local CA and click **Sign Request**.

6. Select **Server** as **Certificate Purpose** and paste the certificate request into the **Certificate Request** field, as shown below.



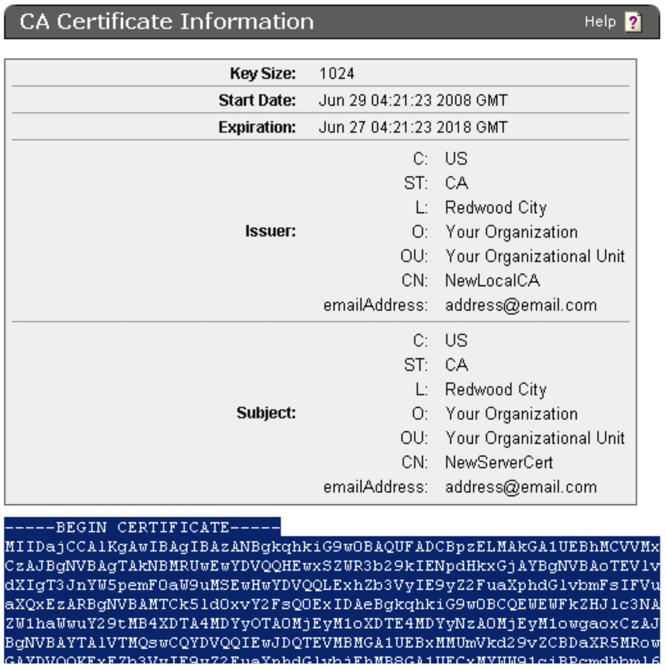
Sign with Certificate Authority:	NewLocalCA (maximum 3649 days)
Certificate Purpose:	[✓ Server] Client
Certificate Duration (days):	3649

Certificate Request:

```
-----BEGIN CERTIFICATE REQUEST-----
MIIB6zCCAVQCAQAwgaoxFjAUBgNVBAMTDU5ld1NlcnZlckNlcnQxGjAYBgNVBAoT
EVlvdXIgT3JnYW5pemF0aW9uMSEwHwYDVQQLExhZb3VyIE9yZ2FuaXphdGlvbmFs
IFVuaXQxFTATBgNVBAcTDFJlZHdvb2QgQ2l0eTELMAkGA1UECBMCQ0ExCzAJBgNV
BAYTAlVTMSAwHgYJKoZIhvcNAQkBFhFhZGRyZXNzQGVtYWlsLmNvbTCBnzANBgkq
hkiG9w0BAQEFAAOBjQAwgYkCgYEA0xJ5a6iOLW9K98eVmkx/k9I6DcHBSVsIO6is
GmYIte74fSoGC0cOTZ3hYL5BReICAOgDsISp47xgv4G9pLHdkHSj+JXebxHLXg1X
zY5OidD/EdB+ptj7XQwzzMewqKVSRIjX/9med3a28ObTf+KinKG6YO9kCTF114ib
R6Gre5MCAwEAAaAAMA0GCSqGSIb3DQEBBQUAA4GBAClKzK7HiHA3KYCqdcBiyn1m
ken9L/updosp/gy68Yv2Lx1ngbLyAFvze+eTNkMLX7ru9sGyuPwytpoXiOf8cdeD
ux1J/PPtOtXDN8oEQ23ZpcHTHZMY49fZvScWVA75gvTdRZVzk9gTVoS4KBboz+tz
```

7. Click **Sign Request**. This will take you to the CA Certificate Information section.

8. Copy the actual certificate (highlighted below). Include the header and footer.



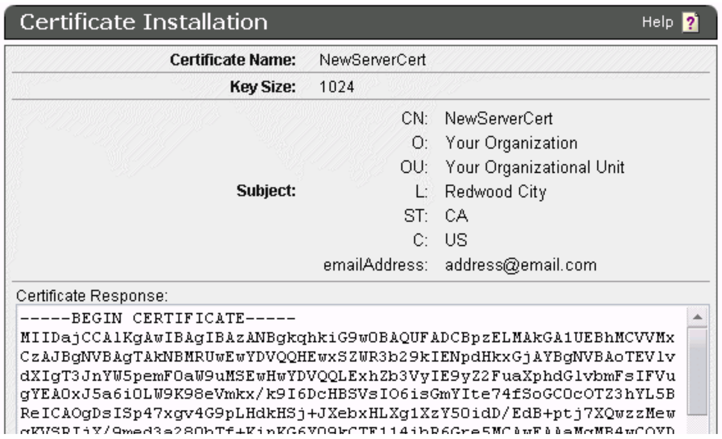
Key Size:		1024
Start Date:		Jun 29 04:21:23 2008 GMT
Expiration:		Jun 27 04:21:23 2018 GMT
Issuer:	C:	US
	ST:	CA
	L:	Redwood City
	O:	Your Organization
	OU:	Your Organizational Unit
	CN:	NewLocalCA
	emailAddress:	address@email.com
Subject:	C:	US
	ST:	CA
	L:	Redwood City
	O:	Your Organization
	OU:	Your Organizational Unit
	CN:	NewServerCert
	emailAddress:	address@email.com

```
-----BEGIN CERTIFICATE-----
MIIDajCCAlKgAwIBAgIBAzANBgkqhkiG9w0BAQUFADCBpzELMAkGA1UEBhMCVVMx
CzAJBgNVBAgTAkNBMRUwEwYDVQQHEwxSZWR3b29kIENpdHkxGjAYBgNVBAoTEVlv
dXIgT3JnYW5pemF0aW9uMSEwHwYDVQQLExhZb3VyIE9yZ2FuaXphdGlvbmFsIFVu
aXQxEzARBgNVBAMTCk5ld0xvY2FsQ0ExIDAeBgkqhkiG9w0BCQEWEWFkZHJlc3NA
ZW1haWwuY29tMB4XDTA4MDYyOTA0MjEyM1oXDTE4MDYyNzA0MjEyM1owgaoxCzAJ
BgNVBAYTAlVTMQswCQYDVQQIEwJDQTEVMBMGA1UEBxMMUmVkd29vZCBDaXR5MRow
```

9. Navigate back to the Certificate List section (Security, CAs & SSL Certificates, SSL Certificates). Select your certificate request and click **Properties**.

10. Click **Install Certificate**.

11. Paste the actual certificate, as shown below. Click **Save**.


Certificate Installation

Certificate Name: NewServerCert

Key Size: 1024

Subject:
CN: NewServerCert
O: Your Organization
OU: Your Organizational Unit
L: Redwood City
ST: CA
C: US
emailAddress: address@email.com

Certificate Response:

```
-----BEGIN CERTIFICATE-----
MIIDajCCAlKgAwIBAgIBAzANBgkqhkiG9w0BAQUFADCBpzELMAkGA1UEBhMCVVMx
CzAJBgNVBAgTAkNBMRUwEwYDVQQHEwxSZWR3b29kIENpdHkxGjAYBgNVBAoTEVlv
dXIgT3JnYW5pemF0aW9uMSEwHwYDVQQLExhZb3VyIE9yZ2FuaXphdGlvbmFsIFVu
gYEA0xJ5a6iOLW9K98eVmkx/k9I6DcHBSVsIO6isGmYIte74fSoGCOcOTZ3hYL5B
ReICAOgDsISp47xgv4G9pLHdkHSj+JXebxHLXg1XzY5OidD/EdB+ptj7XQwzzMew
```

The Certificate List section will now indicate that NewServerCert is an active certificate.

12. Navigate to the Cryptographic Key Server Settings section (Device, Key Server, Key Server). Click **Edit**.
13. Select **Use SSL** and select your new server certificate in the **Server Certificate** field. Click **Save**.
14. Navigate back to the Local Certificate Authority List section (Security, CAs & SSL Certificates, Local CAs). Select your new CA and click **Download**. Place the CA certificate in a secure directory on your client.
15. Update the following parameters in your `IngrianNAE.properties` file:
    - `Protocol=ssl`
    - `CA_File=<path to CA cert>\localca.crt`

    NOTE: Whenever you update the properties file, you must copy the `IngrianNAE.properties` file to all the nodes and restart the database for the changes to take effect.

SSL with Client Certificate Authentication Overview

This SSL implementation requires that both the server and the client produce certificates. Each certificate is signed by a trusted CA known to both the server and the client. Most likely, you will use one CA to sign both certificates. During the SSL handshake, the certificates are exchanged. Both the client and the server use the CA certificate to validate one another's certificate, thus authenticating the other party.

NOTE: For more information about setting up SSL, see "SSL Overview" on page 22.

To enable client certificate authentication, *you must first successfully configure SSL*. Then, you must make additional configuration changes to the client and server.

TIP ! SafeNet recommends that you increase security *only after* confirming network connectivity. You should establish an SSL connection before enabling Client Certificate Authentication. Otherwise, an SSL configuration mistake could interfere with your Client Certificate Authentication setup and complicate the troubleshooting process.

To configure the client, you must:

1. Create a client certificate.

This may involve the following steps:

a. Generating a Client Certificate Request with OpenSSL.

b. Signing a Certificate Request and Downloading the Certificate.

You can create a certificate request using OpenSSL utility. You can then sign the request with the local CA on the SafeNet KeySecure appliance. Once signed, the certificate request becomes a valid certificate.

If you are not using a local CA, consult your CA documentation for instructions on signing requests and exporting certificates.

2. Update `IngrianNAE.properties` file as follows:

   - `Cert_File=<location and name of the client certificate>`
   - `Key_File=<location and name of the client’s key file>`
   - `Passphrase=<the passphrase used to unlock the client’s key file>`

   NOTE: Keep the client certificate file and the associated key file in same directory on all the nodes. Recommended location is `/usr/local/ingrian` directory on all the nodes. Also, `tdatuser` and `tdatudf` should have ownership of the files and the directory containing the files.

   NOTE: Whenever you update the properties file, you must copy the `IngrianNAE.properties` file to all the nodes and restart the database for the changes to take effect.

To configure the server, you must:

1. Place a copy of the CA certificate on your server.

   This may involve the following steps:

   a. Installing a CA Certificate on the Server.

   b. Adding a CA to a Trusted CA List Profile.

2. Update the Authentication Settings section on the SafeNet KeySecure Management Console (Device, Key Server, Key Server).

   a. Navigate to the Cryptographic Key Server Configuration page (**Device**, **Key Server**, **Key Server**).

   b. Select **NAE-XML** as Protocol.

   c. Click **Properties** to access the Authentication Settings section.

   d. Click **Edit** to edit details.

      You need to select either **Used for SSL session only** or **Used for SSL session and username (most secured)** in the **Client Certificate Authentication** field. The profile listed in the **Trusted CA List Profile** field must include the CA used to sign the client certificate. You can update the other fields as needed.

   e. Click **Save**.

SSL with Client Certificate Authentication Procedures

This section describes the procedures you need to follow when configuring SSL with Client Certificate Authentication. It explains the following processes:

- Generating a Client Certificate Request with OpenSSL
- Signing a Certificate Request and Downloading the Certificate
- Installing a CA Certificate on the Server
- Adding a CA to a Trusted CA List Profile

Generating a Client Certificate Request with OpenSSL

To generate a client certificate request:

1. Open a Command Line Terminal and navigate to the directory where the Certificate Request Generator utility (OpenSSL) is located.
2. Generate an RSA key and a client certificate request using the following command:

```
openssl req -out clientreq -newkey rsa:1024 -keyout clientkey
```

NOTE: The certificate request and private key will both be created in the working directory by default. You can generate them in another directory by including a location in the request and key names. For example, to create them in the */usr/local/ingrian* folder, use the following command:

```
openssl req -out /usr/local/ingrian/clientreq -newkey rsa:1024 -keyout /usr/local/
ingrian/clientkey
```

NOTE: Keep the client certificate file and the associated key file in same directory on all the nodes. Recommended location is `/usr/local/ingrian` directory on all the nodes. Also, `tdatuser` and `tdatudf` should have ownership of the files and the directory containing the files.

The key generation process then requests the following data:

- A PEM passphrase to encode the private key.

The passphrase that encodes the private key is the first passphrase you provide after issuing the command above. This will be the `Passphrase` parameter in the `IngrianNAE.properties` file.

- The distinguished name.

The distinguished name is a series of fields whose values are incorporated into the certificate request. These fields include country name, state or province name, locality name, organization name, organizational unit name, common name, e-mail address, surname, user ID, and IP address.

NOTE: For more information about deriving user names and authenticating client IP addresses, see "Authentication Overview" in the *SafeNet KeySecure Appliance User Guide*.

If you will derive the user name from the client certificate, be sure to enter a value in the appropriate field when prompted.

If you will require client certificates to contain a source IP address, be sure to enter the IP address when prompted.

- A challenge password.

This challenge password is NOT used in the SafeNet KeySecure environment.

- An optional company name.

Signing a Certificate Request and Downloading the Certificate

This section describes how to sign a certificate request with a local CA and then download the certificate. You must download the certificate *immediately* after it is signed by the CA.

To sign a certificate request with a local CA:

1. Open the certificate request in a text editor.
2. Copy the text of the certificate request. The copied text must include the header (-----BEGIN CERTIFICATE REQUEST-----) and the footer (-----END CERTIFICATE REQUEST-----).
3. Log on to the SafeNet KeySecure appliance as an administrator with Certificate Authorities access control.
4. Navigate to the Local Certificate Authority List (Security, CAs & SSL Certificates, Local CAs). Select the local CA and click **Sign Request** to access the Sign Certificate Request section.
5. Modify the fields as shown:
   - **Sign with Certificate Authority** - Select the CA that signs the request.
   - **Certificate Purpose** - Select **Client**.
   - **Certificate Duration (days)** - Enter the life span of the certificate.
   - **Certificate Request** - Paste all text from the certificate request, including the header and footer.
6. Click **Sign Request**. This will take you to the CA Certificate Information section.
7. Click the **Download** button to save the certificate on your local machine. You should place the certificate in a secure location and modify access appropriately.

NOTE: Use the `Cert_File` parameter in the `IngrianNAE.properties` file to indicate the name and location of the client certificate. You must copy the `IngrianNAE.properties` file to all the nodes after any modification and must restart the database for the changes to take effect.

Installing a CA Certificate on the Server

If the client certificate was signed by a non-local CA, you must install the CA certificate on the SafeNet KeySecure appliance. To install a CA Certificate:

1. Log on to the SafeNet KeySecure appliance as an administrator with Certificate Authorities access control.
2. Navigate to the Install CA Certificate section on the Certificate and CA Configuration page (Security, CAs & SSL Certificates, Known CAs).
3. Enter the Certificate Name.
4. Paste all text from the certificate in the Certificate field, including the header and footer.
5. Click the **Install** button.

Adding a CA to a Trusted CA List Profile

To add the CA that signed the client certificate to the Trusted CA List Profile:

1. Log on to the SafeNet KeySecure appliance as an administrator with Certificate Authorities access control.

2. Navigate to the Trusted Certificate Authority List Profiles section on the Certificate and CA Configuration page (Security, CAs & SSL Certificates, Trusted CA Lists).
3. Select the profile to which you want to add the CA.
4. Click the **Properties** button.
5. Click the **Edit** button in the Trusted Certificate Authority List section.
6. Select **CA** in the **Available CAs** field and click the **Add** button. This moves your CA from the Available CAs field to the Trusted CAs field.
7. Click the **Save** button.

NOTE: To enable SSL with Client Certificate Authority, the Profile containing the CA that signed the client certificate must be selected as the **Trusted CA List Profile** on the Authentication Settings section.

SSL with Client Certificate Authentication Walkthrough for SafeNet KeySecure Clients

This walkthrough assumes the following:

- You have read the SSL with Client Certificate Authentication overview section.
- You have successfully completed the SSL Walkthrough for SafeNet KeySecure Clients. *You must use the Local CA created in that walkthrough.* The instructions below assume that the client and server certificates were signed by the same local CA.

There are a few different ways that you could configure SSL with Client Certificate Authentication. For example, you can use a CA that does not reside on the SafeNet KeySecure appliance or you can create a new CA. This walkthrough makes such decisions for you. By following these instructions, you will:

- Create a Client Certificate Request using OpenSSL utility.
- Create a Client Certificate by signing the Client Certificate Request with the Local CA on the SafeNet KeySecure appliance.
- Add the Local CA to the Trusted CA List.

Once you have completed and understood this walkthrough, you might decide to alter some steps to better fit your organization's policies.

To configure client certificate authentication:

1. Open a Command Line Terminal and navigate to the directory where the Certificate Request Generator utility (OpenSSL) is located.
2. Generate an RSA key and a client certificate request using the following command:
   ```
   openssl req -out ssl\clientreq -newkey rsa:1024 -keyout ssl\clientkey
   ```
   The key generation process then requests the following data:
   - A PEM passphrase to encode the private key.

The passphrase that encodes the private key is the first passphrase you provide after issuing the command above. This will be the Passphrase parameter in the `IngrianNAE.properties` file.

- The distinguished name.

The distinguished name is a series of fields whose values are incorporated into the certificate request. These fields include country name, state or province name, locality name, organization name, organizational unit name, common name, e-mail address, surname, user ID, and IP address.

NOTE: For more information about deriving NAE user names and authenticating client IP addresses, see "Authentication Overview" in the *SafeNet KeySecure Appliance User Guide*.

If you will derive the NAE user name from the client certificate, be sure to enter a value in the appropriate field when prompted.

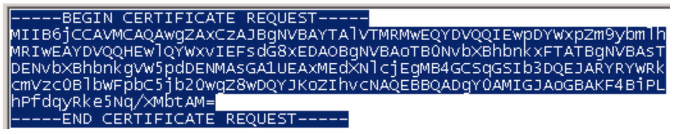
If you will require client certificates to contain a source IP address, be sure to enter the IP address when prompted.

- A challenge password.

This challenge password is NOT used in the SafeNet KeySecure environment.

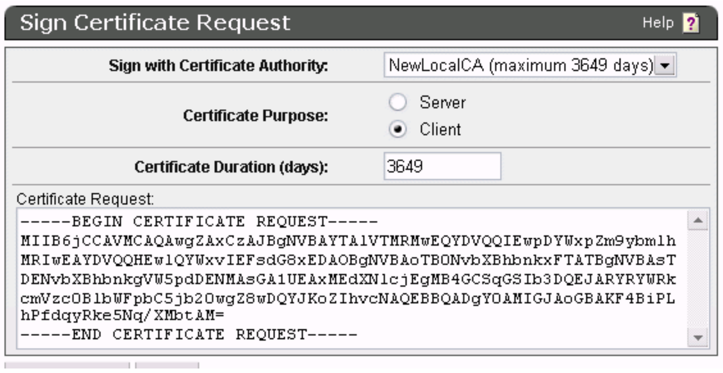
- An optional company name.

3. Open the client certificate request file and copy the actual request (highlighted below). Include the header and footer.



```
-----BEGIN CERTIFICATE REQUEST-----
MIIB6jCCAVMCAQAwgZAxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlh
MRIwEAYDVQQHEwlQYWxvIEFsdG8xEDAOBgNVBAoTB0NvbXBhbnkxFTATBgNVBAsT
DENvbXBhbnkgVW5pdDENMAsGA1UEAxMEdXNlcjEgMB4GCSqGSIb3DQEJARYRYWRk
cmVzc0BlbWFpbC5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAKF4BiPL
hPfdqyRke5Nq/XMbtAM=
-----END CERTIFICATE REQUEST-----
```

4. Log on to the SafeNet KeySecure Management Console as an administrator with Certificate Authorities access control.
5. Navigate to the Local Certificate Authority List section (Security, CAs & SSL Certificates, Local CAs). Select **NewLocalCA** and click **Sign Request**. (NewLocalCA is the CA you created in the SSL Walkthrough.)
6. Select **Client** as **Certificate Purpose** and paste the certificate request into the **Certificate Request** field, as shown below.



Sign with Certificate Authority: NewLocalCA (maximum 3649 days)

Certificate Purpose: Server [✓ Client]

Certificate Duration (days): 3649

Certificate Request:

```
-----BEGIN CERTIFICATE REQUEST-----
MIIB6jCCAVMCAQAwgZAxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlh
MRIwEAYDVQQHEwlQYWxvIEFsdG8xEDAOBgNVBAoTB0NvbXBhbnkxFTATBgNVBAsT
DENvbXBhbnkgVW5pdDENMAsGA1UEAxMEdXNlcjEgMB4GCSqGSIb3DQEJARYRYWRk
cmVzc0BlbWFpbC5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAKF4BiPL
hPfdqyRke5Nq/XMbtAM=
-----END CERTIFICATE REQUEST-----
```

7. Click **Sign Request**. This will take you to the CA Certificate Information section.
8. Click **Download** to download your new client certificate (signed.crt) to your client. Place the certificate in a secure directory on your client.

9. Navigate to the Trusted Certificate Authority List Profiles section (Security, CAs & SSL Certificates, Trusted CA Lists). Select **Default** as **Profile Name** and click **Properties**.
10. Click **Edit** in the Trusted Certificate Authority List section.
11. Select **NewLocalCA** in **Available CAs** and click **Add**. Click **Save**.
12. Update the following parameters in the `IngrianNAE.properties` file:
    - `Cert_File=<path to client cert>\client.crt`
    - `Key_File=<path to client key>\clientkey`
    - `Passphrase=<the passphrase used to unlock the client’s key file>`

    NOTE: Whenever you update the properties file, you must copy the `IngrianNAE.properties` file to all the nodes and restart the database for the changes to take effect.

13. Return to the SafeNet KeySecure Management Console and navigate to the Authentication Settings section (Device, Key Server, Key Server) and enter the following values:
    - **Client Certificate Authentication**: Used for SSL Session only
    - **Trusted CA List Profile**: Default

    IMPORTANT ! The CA used to sign the client certificate must be a member of the **Trusted CA List Profile**.

6 Standard Encryption and Decryption

This chapter contains the following topics:

NOTE: Before encrypting/decrypting any data, ensure that `tdatuser` and `tdatudf` has the ownership of all the files in `/usr/local/ingrian` directory.

Supported Data Types

SafeNet ProtectDB for Teradata supports the following data types for standard encryption/decryption:

- `CHAR`
- `VARCHAR`
- `INT`
- `BYTEINT`
- `SMALLINT`
- `DATE`
- `TIME`
- `TIMESTAMP`

NOTE: For CHAR and VARCHAR data types, maximum data length of 256 is supported.

Encrypting and Decrypting Using a Key Name for Standard Encryption

The following user defined functions enable you to encrypt and decrypt data using a specific key, algorithm, mode of operation, padding method and IV for standard encryption/decryption:

```
SFNT_ENCRYPT_CHAR
SFNT_DECRYPT_CHAR
SFNT_ENCRYPT_INT
SFNT_DECRYPT_INT
SFNT_ENCRYPT_SMLINT
SFNT_DECRYPT_SMLINT
SFNT_ENCRYPT_BINT
SFNT_DECRYPT_BINT
SFNT_ENCRYPT_DAT
SFNT_DECRYPT_DAT
SFNT_ENCRYPT_TIME
SFNT_DECRYPT_TIME
SFNT_ENCRYPT_TIMESTMP
SFNT_DECRYPT_TIMESTMP
```

SFNT_ENCRYPT_CHAR

Use `SFNT_ENCRYPT_CHAR` to encrypt a character string using the key, fully-qualified algorithm and IV passed to the function.

```
VARBYTE(256) SFNT_ENCRYPT_CHAR
   USER_NAME  VARCHAR(256)
   INPUT_DATA VARCHAR(256)
   KEY_NAME   VARCHAR(256)
   ALG_NAME   VARCHAR(256)
   IV         VARBYTE(16)
```

NOTE: The parameters for encryption operation using the user defined functions provided in SafeNet ProtectDB for Teradata are same; and user should provide data in the `Input_Data` parameter corresponding to the data type.

Parameter	Description
USER_NAME	Pass the value `USER` in this field. Authentication is done based on the logged in database user only.
INPUT_DATA	The data to be encrypted. You can pass a string or a column name to encrypt an entire column. The string can be up to 256 characters long. **NOTE:** Ensure to provide the input value with correct format for corresponding data type.
KEY_NAME	The name of the key used to encrypt. The logged in database user must be mapped to a KeySecure user with encryption privileges for this key.
ALG_NAME	Contains the fully-qualified algorithm name, which includes the algorithm name, mode of operation, and padding method. For example, AES/CBC/PKCS5Padding.
IV	Contains the initialization vector needed to encrypt. The IV value is 32 bit hex-encoded byte array, such as '12345678123456781234567812345678'XBV.

SFNT_DECRYPT_CHAR

Use `SFNT_DECRYPT_CHAR` to decrypt a character string using the key, fully-qualified algorithm and IV passed to the function.

```
VARBYTE(256) SFNT_DECRYPT_CHAR
   USER_NAME   VARCHAR(256)
   INPUT_DATA  VARCHAR(256)
   KEY_NAME    VARCHAR(256)
   ALG_NAME    VARCHAR(256)
   IV          VARBYTE(16)
```

NOTE: The parameters for decryption operation using the user defined functions provided in SafeNet ProtectDB for Teradata are same; and user should provide data in the `Input_Data` parameter corresponding to the data type.

Parameter	Description
USER_NAME	Pass the value `USER` in this field. Authentication is done based on the logged in database user only.
INPUT_DATA	The ciphertext to be decrypted. You can pass a string or a column name. The ciphertext can be up to 256 characters long.
KEY_NAME	The name of the key used to decrypt. The logged in database user must be mapped to a KeySecure user with decryption privileges for this key.

Parameter	Description
ALG_NAME	Contains the fully-qualified algorithm name, which includes the algorithm name, mode of operation, and padding method. For example, AES/CBC/PKCS5Padding.
IV	Contains the initialization vector needed to decrypt. The IV value is 32 bit hex-encoded byte array, such as '12345678123456781234567812345678'XBV.

Examples of Standard Encryption/Decryption

The following code samples give examples of encrypting and decrypting columns and strings for `CHAR` data type.

Encrypting a Column and Storing in Table

This example encrypts data in table XACT and stores it in table XACT_enc.

To create table XACT, execute the following code:

```
CREATE TABLE XACT(aint int, CCNUM char(16), TRANS char(32));
INSERT INTO XACT VALUES(1, '9999123499991234','99991234999912349999123499991234');
INSERT INTO XACT VALUES(2, '8766123487661234','87661234876612348766123487661234');
INSERT INTO XACT VALUES(3, '1111123411111234','11111234111112341111123411111234');
INSERT INTO XACT VALUES(4, '7654321176543211','76543211765432117654321176543211');
INSERT INTO XACT VALUES(5, '1010123410101234','10101234101012341010123410101234');
SELECT * FROM XACT ORDER BY AINT;
.SET foldline ON ALL;
```

To create table XACT_enc, execute the following code:

```
CREATE TABLE XACT_enc(aint int, CCNUMenc varbyte(50), TRANSenc varbyte(50));
```

To encrypt columns from table XACT and store the resulting ciphertext in XACT_enc, execute the following code:

```
INSERT into XACT_enc
SELECT AINT,
INGRIAN.SFNT_ENCRYPT_CHAR(USER, CCNUM, 'dbuaes192', 'AES/CBC/PKCS5Padding',
'12345678123456781234567812345678'XBV) CCNUM,
INGRIAN.SFNT_ENCRYPT_CHAR(USER, TRANS, 'dbuaes256', 'AES/CBC/PKCS5Padding',
'12345678123456781234567812345678'XBV) TRANS
FROM XACT;
SELECT * FROM XACT_enc order by AINT;
```

Decrypting a Column and Storing in Table

This example decrypts data from table XACT_enc and stores it in table XACT_dec.

To decrypt columns from table XACT_enc and store the resulting plaintext in XACT_dec, execute the following code:

```
CREATE TABLE XACT_dec (aint int, CCNUM char(250), TRANS char(250));
INSERT into XACT_dec SELECT AINT,
INGRIAN.SFNT_DECRYPT_CHAR(USER,CCNUMenc,
'dbuaes192','AES/CBC/PKCS5Padding','12345678123456781234567812345678'XBV)
CCNUMdec,
INGRIAN.SFNT_DECRYPT_CHAR(USER, TRANSenc, 'dbuaes256', 'AES/CBC/
PKCS5Padding','12345678123456781234567812345678'XBV) TRANSdec FROM XACT_enc;
SELECT * FROM XACT_dec order by AINT;
```

Encrypting a String

The following code encrypts a string `abcdef` with key `dbuaes128` using AES/CBC/PKCS5Padding:

```
SELECT SFNT_ENCRYPT_CHAR(USER,'abcdef', 'dbuaes128', 'AES/CBC/PKCS5Padding',
'12345678123456781234567812345678'XBV);
```

Decrypting a String

The following code decrypts the string

```
'CA4AC3965BEFEFED0CBA9AE6ED9F8DF6'
```

```
SELECT SFNT_DECRYPT_CHAR(USER,'CA4AC3965BEFEFED0CBA9AE6ED9F8DF6'XBV, 'dbuaes128',
'AES/CBC/PKCS5Padding', '12345678123456781234567812345678'XBV);
```

7 FPE Encryption and Decryption

The FPE algorithm allows the user to perform encryption on well formatted data without affecting its format post encryption. This chapter contains the following topics:

NOTE: Before encrypting/decrypting any data, ensure that `tdatuser` and `tdatudf` has the ownership of all the files in `/usr/local/ingrian` directory. To use the `ALPHANUMFPE` UDFs, set the `Symmetric_Key_Cache_Enabled = tcp_ok` in the `IngrianNAE.properties` file.

Supported Data Types

SafeNet ProtectDB for Teradata supports the following data types for FPE encryption/decryption:

- `CHAR`
- `VARCHAR`
- `INT`
- `BYTEINT`
- `SMALLINT`

NOTE: For FPE operation, it is recommended to use minimum data length of two bytes. And for CHAR and VARCHAR data types, maximum data length of 256 is supported.

Encrypting and Decrypting Using a Key Name for FPE Encryption

The following user defined functions enable you to encrypt and decrypt well formated data using a specific key, algorithm, mode of operation, padding method, IV, tweak algorithm and tweak data for format preserving encryption (FPE) operations:

UDFs	Description
`SFNT_ENCRYPT_INTFPE` `SFNT_DECRYPT_INTFPE`	These UDFs use CARD10 cardinality (algorithm `FPE/AES/CARD10`) for digits in the range 0 - 9. **NOTE:** Supported in both local and remote mode.
`SFNT_ENCRYPT_CHARFPE` `SFNT_DECRYPT_CHARFPE`	These UDFs use CARD10 cardinality (algorithm `FPE/AES/CARD10`) for digits in the range 0 - 9. Input data with alphanumeric and special characters are allowed. However, during encryption/decryption both the alphabets and special characters are retained and only the numeric part of the input data will be encrypted/decrypted. **NOTE:** Supported in both local and remote mode.
`SFNT_ENCRYPT_ALPHANUMFPE` `SFNT_DECRYPT_ALPHANUMFPE`	These UDFs use CARD62 cardinality (algorithm `FPE/AES/CARD62`) for digits in the range 0 - 9; and alphabets in the range a - z and A - Z. Input data with alphanumeric and special characters are allowed. The special characters are retained after encryption/decryption. **NOTE:** Supported in local mode only. Also, set the parameter `Symmetric_Key_Cache_Enabled = tcp_ok` in the `IngrianNAE.properites` file in `/usr/local/ingrian` directory for CARD62 cardinality. Ensure to replace this file in all the nodes after the modification.

SFNT_ENCRYPT_CHARFPE

Use `SFNT_ENCRYPT_CHARFPE` to encrypt a character string using the key, fully-qualified algorithm, IV, tweak data and tweak algorithm passed to the function.

```
VARBYTE(256) SFNT_ENCRYPT_CHARFPE
   USER_NAME  VARCHAR(256)
   INPUT_DATA VARCHAR(256)
   KEY_NAME   VARCHAR(256)
   ALG_NAME   VARCHAR(256)
   IV         VARBYTE(112)
   TWEAK_ALG  VARCHAR(256)
   TWEAK_DATA VARCHAR(256)
```

NOTE: The parameters of the user defined functions used for FPE encryption operation are same; and user should provide data in the `Input_Data` parameter corresponding to the data type and algorithm name in the `ALG_NAME` parameter as per the supported cardinality.

Parameter	Description
USER_NAME	Pass the value `USER` in this field. Authentication is done based on the logged in database user only.
INPUT_DATA	The data to be encrypted. You can pass a string or a column name to encrypt an entire column. The string can be up to 256 characters long. **NOTE:** Ensure to provide the input value with correct format for corresponding data type.
KEY_NAME	The name of the key used to encrypt. The logged in database user must be mapped to a KeySecure user with encryption privileges for this key. **NOTE:** Non-versioned AES Keys are used. Key versions not supported.
ALG_NAME	Contains the fully-qualified algorithm name: `FPE/AES/CARD10`. **NOTE:** Use the algorithm `FPE/AES/CARD10` or `FPE/AES/CARD62` as applicable for the UDF.

Parameter	Description
IV	Contains the initialization vector needed to encrypt. FPE accepts a HEX encoded MAXb integer. IV is used only if the length of data exceeds MAXb (block size: 56 bytes for `CARD10` and 32 bytes for `CARD62`). FPE breaks long data into MAXb integer blocks and uses block chaining algorithm similar to CBC mode to perform encryption. • a 56 bytes IV in hex encoded form having a cardinality 10 when data size is >56 bytes. A valid value of IV can be a 112 characters hex encoded s-integers (0-9) "0401030003040604090301030705020505030507040108080102020704020702010304070400090105020603000002020906070004010200”. • a 32 bytes IV in hex encoded form having a cardinality 62 when data size is >32 bytes. A valid value of IV can be a 64 characters hex encoded s-integers (0-9) "040103000304060409030103070502050503050704010808010202070420702”. **NOTE:** Pass a null value if the data length is less than or equal to the maximum block size. Otherwise, IV is not used even if it is provided by user. **NOTE:** The IV will be used only in case the data to be encrypted (excluding special characters/alphabets as per the cardinality used) is greater than 56 bytes in case of CARD10 and 32 bytes in case of CARD62.
Tweak Algorithm	Used to let caller specify a hashing algorithm to be applied to specified tweak data beforehand. Valid value: • NONE • SHA1 • SHA256
Tweak Data	Tweak data uses the tweakable cipher concept to protect against statistical attacks due to potentially small input/output space. It accepts any ASCII value for SHA1 and SHA 256 and any valid hex encoded value for “NONE” like "1111111111111111". If, tweak data algorithm is “NONE”, the value must be HEX encoded string representing 64 bit long (hence, HEX encoding will consume 16 characters.). If tweak data algorithm represents a SHA1/SHA256 argument then the tweak data need not be hex encoded string but any ASCII string.

SFNT_DECRYPT_CHARFPE

Use `SFNT_DECRYPT_CHARFPE` to decrypt a character string using the key, fully-qualified algorithm, IV, tweak algorithm and tweak data passed to the function.

```
VARBYTE(256) SFNT_DECRYPT_CHARFPE
   USER_NAME  VARCHAR(256)
   INPUT_DATA VARCHAR(256)
   KEY_NAME   VARCHAR(256)
   ALG_NAME   VARCHAR(256)
   IV         VARBYTE(112)
   TWEAK_ALG  VARCHAR(256)
   TWEAK_DATA VARCHAR(256)
```

NOTE: The parameters for decryption operation using the user defined functions provided in SafeNet ProtectDB for Teradata are same; and user should provide data in the `Input_Data` parameter corresponding to the data type and algorithm name in the `ALG_NAME` parameter as per the supported cardinality.

Parameter	Description
USER_NAME	Pass the value `USER` in this field. Authentication is done based on the logged in database user only.
INPUT_DATA	The ciphertext to be decrypted. You can pass a string or a column name to encrypt an entire column. The string can be up to 256 characters long.
KEY_NAME	The name of the key used to decrypt. The logged in database user must be mapped to a KeySecure user with decryption privileges for this key. **NOTE:** Non-versioned AES Keys are used. Key versions not supported.
ALG_NAME	Contains the fully-qualified algorithm name: `FPE/AES/CARD10`. **NOTE:** Use the algorithm `FPE/AES/CARD10` or `FPE/AES/CARD62` as applicable for the UDF.

Parameter	Description
IV	Contains the initialization vector needed to decrypt. FPE accepts a HEX encoded MAXb integer. IV is used only if the length of data exceeds MAXb (block size: 56 bytes for `CARD10` and 32 bytes for `CARD62`). FPE breaks long data into MAXb integer blocks and uses block chaining algorithm similar to CBC mode to perform encryption. • a 56 bytes IV in hex encoded form having a cardinality 10 when data size is >56 bytes. A valid value of IV can be a 112 characters hex encoded s-integers (0-9) "04010300030406040903010307050205050305070401080801020207040207020103040704000901050206030000020209060700040102 00”. • a 32 bytes IV in hex encoded form having a cardinality 62 when data size is >32 bytes. A valid value of IV can be a 64 characters hex encoded s-integers (0-9) "0401030003040604090301030705020505030507040108080102020704020702”. **NOTE:** Pass a null value if the data length is less than or equal to the maximum block size. Otherwise, IV is not used even if it is provided by user. **NOTE:** The IV will be used only in case the data to be decrypted (excluding special characters/alphabets as per the cardinality used) is greater than 56 bytes in case of CARD10 and 32 bytes in case of CARD62.
Tweak Algorithm	Used to let caller specify a hashing algorithm to be applied to specified tweak data beforehand. Valid value: • NONE • SHA1 • SHA256
Tweak Data	Tweak data uses the tweakable cipher concept to protect against statistical attacks due to potentially small input/output space. It accepts any ASCII value for SHA1 and SHA 256 and any valid hex encoded value for “NONE” like "1111111111111111". If, tweak data algorithm is “NONE”, the value must be HEX encoded string representing 64 bit long (hence, HEX encoding will consume 16 characters.). If tweak data algorithm represents a SHA1/SHA256 argument then the tweak data need not be hex encoded string but any ASCII string.

Examples of FPE Encryption/Decryption for CARD10 and CARD62

The following code samples give examples of encrypting and decrypting columns and strings for `CHAR` data type for FPE encryption/decryption.

Encrypting a Column and Storing in Table

This example encrypts data in table XACT_FPE and stores it in table XACT_FPE_enc.

To create table XACT_FPE, execute the following code:

```
CREATE TABLE XACT_FPE(aint int, CCNUM char(16), TRANS char(32));
INSERT INTO XACT_FPE
VALUES(1,'9999123499991234','99991234999912349999123499991234');
INSERT INTO XACT_FPE
VALUES(2,'8766123487661234','87661234876612348766123487661234');
INSERT INTO XACT_FPE
VALUES(3,'1111123411111234','11111234111112341111123411111234');
INSERT INTO XACT_FPE VALUES(4,'7654321176543211',
'76543211765432117654321176543211');
INSERT INTO XACT_FPE
VALUES(5,'1010123410101234','10101234101012341010123410101234');
SELECT * FROM XACT_FPE ORDER BY AINT;
.SET foldline ON ALL;
```

To create table XACT_FPE_enc, execute the following code:

```
CREATE TABLE XACT_FPE_enc (aint int,CCNUMenc varchar(16), TRANSenc varchar(32));
```

To encrypt columns from table XACT_FPE and store the resulting ciphertext in XACT_FPE_enc, execute the following code:

```
INSERT into XACT_FPE_enc SELECT AINT,
INGRIAN.SFNT_ENCRYPT_CHARFPE(USER,CCNUM,'dbuaes192','FPE/AES/CARD10',null,'SHA1',
'12345') CCNUM, INGRIAN.SFNT_ENCRYPT_CHARFPE(USER,TRANS , 'dbuaes256', 'FPE/AES/
CARD10',null,'SHA1', '12345') TRANS FROM XACT_FPE;
SELECT * FROM XACT_FPE_enc order by AINT;
```

Decrypting a Column and Storing in Table

This example decrypts data from table XACT_FPE and stores it in the table XACT_FPE_dec.

To decrypt columns from table XACT_FPE_enc and store the resulting plaintext in XACT_FPE_dec, execute the following code:

```
CREATE TABLE XACT_FPE_dec(aint int, CCNUM char(16), TRANS char(32));
INSERT into XACT_FPE_dec SELECT AINT, INGRIAN.SFNT_DECRYPT_CHARFPE(USER,CCNUMenc,
'dbuaes192','FPE/AES/CARD10',null,'SHA1', '12345') CCNUMdec,
INGRIAN.SFNT_DECRYPT_CHARFPE(USER, TRANSenc, 'dbuaes256','FPE/AES/
CARD10',null,'SHA1', '12345') TRANSdec FROM XACT_FPE_enc;

SELECT * FROM XACT_FPE_dec order by AINT;
```

Encrypting a String for CARD10

The following code encrypts a string ‘`ABC:123456`’ with key `dbuaes128` using `FPE/AES/CARD10`:

```
SELECT SFNT_ENCRYPT_CHARFPE(USER,'ABC:123456','dbuaes128','FPE/AES/
CARD10',null,'SHA1','12345');
```

The output is `ABC:657359.`

Decrypting a String for CARD10

The following code decrypts the string `'ABC:657359'`

```
SELECT SFNT_DECRYPT_CHARFPE(USER,'ABC:657359','dbuaes128','FPE/AES/
CARD10',null,'SHA1','12345');
```

The output is `'ABC:123456'`

Encrypting a String for CARD62

The following code encrypts a string ‘`Test_Mail123@Domian.com`’ with key `dbuaes128` using `FPE/AES/CARD62`:

```
SELECT SFNT_ENCRYPT_ALPHANUMFPE(USER,'Test_Mail123@Domian.com','dbuaes128','FPE/
AES/CARD62',null,'SHA1','12345');
```

The output is ‘`KN1c_LpykD7T@1yikeN.dpo’.`

Decrypting a String for CARD62

The following code decrypts the string `'KN1c_LpykD7T@1yikeN.dpo'`

```
SELECT SFNT_DECRYPT_ALPHANUMFPE(USER,'KN1c_LpykD7T@1yikeN.dpo','dbuaes128','FPE/
AES/CARD62',null,'SHA1','12345');
```

The output is `'Test_Mail123@Domian.com'.`