THALES

Vormetric Data Security Platform

Vormetric Protection for Teradata Database

Installation and Reference Guide

Version 6.4.0

Documentation v1

April 7, 2020

Vormetric Data Security Platform
Vormetric Protection for Teradata Database (VPTD) Version 6.4.0
Installation and Reference Guide v1
April 7, 2020,

Copyright 2009 – 2020. Thales eSecurity, Inc. All rights reserved.

NOTICES, LICENSES, AND USE RESTRICTIONS

Vormetric, Thales, and other Thales trademarks and logos are trademarks or registered trademark of Thales eSecurity, Inc. in the United States and a trademark or registered trademark in other countries.

All other products described in this document are trademarks or registered trademarks of their respective holders in the United States and/or in other countries.

The software ("Software") and documentation contains confidential and proprietary information that is the property of Thales eSecurity, Inc. The Software and documentation are furnished under license from Thales and may be used only in accordance with the terms of the license. No part of the Software and documentation may be reproduced, transmitted, translated, or reversed engineered, in any form or by any means, electronic, mechanical, manual, optical, or otherwise.

The license holder ("Licensee") shall comply with all applicable laws and regulations (including local laws of the country where the Software is being used) pertaining to the Software including, without limitation, restrictions on use of products containing encryption, import or export laws and regulations, and domestic and international laws and regulations pertaining to privacy and the protection of financial, medical, or personally identifiable information. Without limiting the generality of the foregoing, Licensee shall not export or re-export the Software, or allow access to the Software to any third party including, without limitation, any customer of Licensee, in violation of U.S. laws and regulations, including, without limitation, the Export Administration Act of 1979, as amended, and successor legislation, and the Export Administration Regulations issued by the Department of Commerce, or in violation of the export laws of any other country.

Any provision of any Software to the U.S. Government is with "Restricted Rights" as follows: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277.7013, and in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause at FAR 52.227-19, and in similar clauses in the NASA FAR Supplement, when applicable. The Software is a "commercial item" as that term is defined at 48 CFR 2.101, consisting of "commercial computer software" and "commercial computer software documentation", as such terms are used in 48 CFR 12.212 and is provided to the U.S. Government and all of its agencies only as a commercial end item. Consistent with 48 CFR 12.212 and DFARS 227.7202-1 through 227.7202-4, all U.S. Government end users acquire the Software with only those rights set forth herein. Any provision of Software to the U.S. Government is with Limited Rights. Thales is Thales eSecurity, Inc. at Suite 710, 900 South Pine Island Road, Plantation, FL 33324.

THALES PROVIDES THIS SOFTWARE AND DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, TITLE, NON-INFRINGEMENT OF THIRD PARTY RIGHTS, AND ANY WARRANTIES ARISING OUT OF CONDUCT OR INDUSTRY PRACTICE. ACCORDINGLY, THALES DISCLAIMS ANY LIABILITY, AND SHALL HAVE NO RESPONSIBILITY, ARISING OUT OF ANY FAILURE OF THE SOFTWARE TO OPERATE IN

ANY ENVIRONMENT OR IN CONNECTION WITH ANY HARDWARE OR TECHNOLOGY, INCLUDING, WITHOUT LIMITATION, ANY FAILURE OF DATA TO BE PROPERLY PROCESSED OR TRANSFERRED TO, IN OR THROUGH LICENSEE'S COMPUTER ENVIRONMENT OR ANY FAILURE OF ANY TRANSMISSION HARDWARE, TECHNOLOGY, OR SYSTEM USED BY LICENSEE OR ANY LICENSEE CUSTOMER. THALES SHALL HAVE NO LIABILITY FOR, AND LICENSEE SHALL DEFEND, INDEMNIFY, AND HOLD THALES HARMLESS FROM AND AGAINST, ANY SHORTFALL IN PERFORMANCE OF THE SOFTWARE, OTHER HARDWARE OR TECHNOLOGY, OR FOR ANY INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS, AS A RESULT OF THE USE OF THE SOFTWARE IN ANY ENVIRONMENT. LICENSEE SHALL DEFEND, INDEMNIFY, AND HOLD THALES HARMLESS FROM AND AGAINST ANY COSTS, CLAIMS, OR LIABILITIES ARISING OUT OF ANY AGREEMENT BETWEEN LICENSEE AND ANY THIRD PARTY. NO PROVISION OF ANY AGREEMENT BETWEEN LICENSEE AND ANY THIRD PARTY SHALL BE BINDING ON THALES.

Protected by U.S. patents:

6,678,828

6,931,530

7,143,288

7,283,538

7,334,124

Contents

PREFACE

This manual describes how to install and implement Vormetric Protection for Teradata Database (VPTD) on your Teradata nodes.

DOCUMENTATION VERSION HISTORY

The following table describes the changes made for each document version.

Documentation Changes

Document Version	Date	Changes
6.4.0	4/07/20	Added support for using SafeNet KeySecure as a key manager (via its ICAPI crypto library) for encryption/decryption in addition to the existing DSM key manager support. Added support for Teradata MultiLoad and FastExport utilities. See "Teradata MultiLoad and FastExport" on page 75. Added several new UDFs to support datatypes INT, BYTEINT, SMALLINT, DATE, TIME, and TIMESTAMP. UDFs for standard CBC encryption and decryption add support for all of these datatypes. UDFs for FPE and FF1 encryption/decryption have been added to support INT, BYTEINT, and SMALLINT datatypes. See "VPTD UDFs" on page 41.
6.3.0	8/27/19	Added support for the FF1 algorithm, including new UDFs "encrypt_ff1()" on page 58 and "decrypt_ff1()" on page 59.
6.1.3	12/7/18	Added content to describe user restriction based on identity.
6.0.2. v1	5/10/18	Updated illustrations. Added new features: Dynamic masking, install, uninstall and upgrade over a TD cluster.
5.2.5 v2	1/25/17	Add integrated installer (vptd-*).
5.2.5 v1	12/16/16	GA release of 5.2.5 document.
5.2.5 Beta v1	9/2/16	Added UDFs for Unicode block cipher and FPE encryption: encrypt_cbc( ), decrypt_cbc( ), encrypt_fpe( ), and decrypt_fpe( ).

ASSUMPTIONS

This documentation assumes knowledge of the Teradata database.

The system administrator must have root permissions for the systems on which Vormetric Protection for Teradata Database software is installed.

GUIDE TO VORMETRIC PROTECTION FOR TERADATA DATABASE DOCUMENTATION

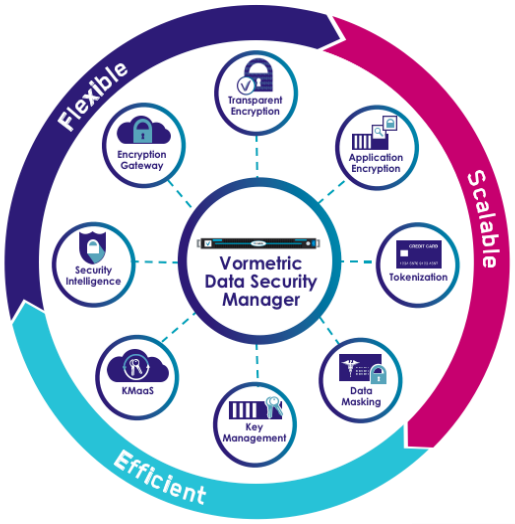
The following related documents are available to registered users on the Thales Support Portal at https://supportportal.thalesgroup.com

1. **Data Security Manager (DSM) Installation and Configuration Guide**. Use this to install the Data Security Manager.
2. **Vormetric Protection for Teradata Database Installation and Reference**. This document. For security professionals who want to protect their Teradata Database.
3. **Vormetric Security Intelligence Configuration Guide**. Use this to integrate your Vormetric Tokenization event logs with the ArcSight ESM, Splunk, or IBM QRadar
4. **Vormetric Data Security (VDS) Platform Event and Log Messages Reference** (~700 pages, 0% pictures). A listing of all the VDS Platform event and log messages with severity, long and short form, and description.

VORMETRIC DATA SECURITY PLATFORM—OVERVIEW

The Vormetric Data Security (VDS) Platform protects data wherever it resides. The platform solves security and compliance issues with encryption, key management, privileged user access control, and security intelligence logging. It protects data in databases, files, and Big Data nodes across public, private, hybrid clouds and traditional infrastructures.

Figure 1: The Vormetric "Solar System"



The platform consists of products that share a common, extensible infrastructure. At the heart of the platform is the Data Security Manager (DSM), which coordinates policies, keys, and the collection of security intelligence, all of which is managed and observed through your browser. In addition to the DSM, the Vormetric Data Security Platform consists of the following products:

- **Vormetric Application Encryption (VAE)** provides a framework to deliver application-layer encryption such as column- or field-level encryption in databases, Big Data, or PaaS applications. VAE is an application encryption library providing a standards-based API to do cryptographic and encryption key management operations into existing corporate applications.
- **Vormetric Key Management (VKM)** centralizes 3rd-party encryption keys and stores certificates securely. It provides standards-based enterprise encryption key management for Transparent Database Encryption (TDE), KMIP-compliant devices, and offers vaulting and inventory of certificates.
- **Vormetric Protection for Teradata Database (VPTD)** provides granular controls to secure assets in Teradata environments. It simplifies the process of using column-level encryption in your Teradata database. It provides documented, standards-based APIs and user-defined functions (UDFs) for cryptographic and key management operations.
- **Vormetric Security Intelligence** provides comprehensive logging combined with Security Information Event Management (SIEM) systems to detect advanced persistent threats and insider threats. In addition, the logs satisfy compliance and regulatory audits.
- **Vormetric Tokenization (VTS)** replaces sensitive data in your database with unique identification symbols called tokens. This reduces the number of places that clear-text

sensitive data reside, and thus reduces the scope of complying with PCI DSS and corporate security policies.

- **Vormetric Transparent Encryption (VTE)** secures any database, file, or volume across your enterprise without changing the applications, infrastructure, or user experience.

SERVICE UPDATES AND SUPPORT INFORMATION

The license agreement that you have entered into to acquire the Thales products ("License Agreement") defines software updates and upgrades, support and services, and governs the terms under which they are provided. Any statements made in this guide or collateral documents that conflict with the definitions or terms in the License Agreement, shall be superseded by the definitions and terms of the License Agreement. Any references made to "upgrades" in this guide or collateral documentation can apply either to a software update or upgrade.

SALES AND SUPPORT

For support and troubleshooting issues:

- https://supportportal.thalesgroup.com
- (888) 343-5773

For Thales Sales:

- http://enterprise-encryption.vormetric.com/contact-sales.html
- sales@thalesesec.net
- (888) 267-3732

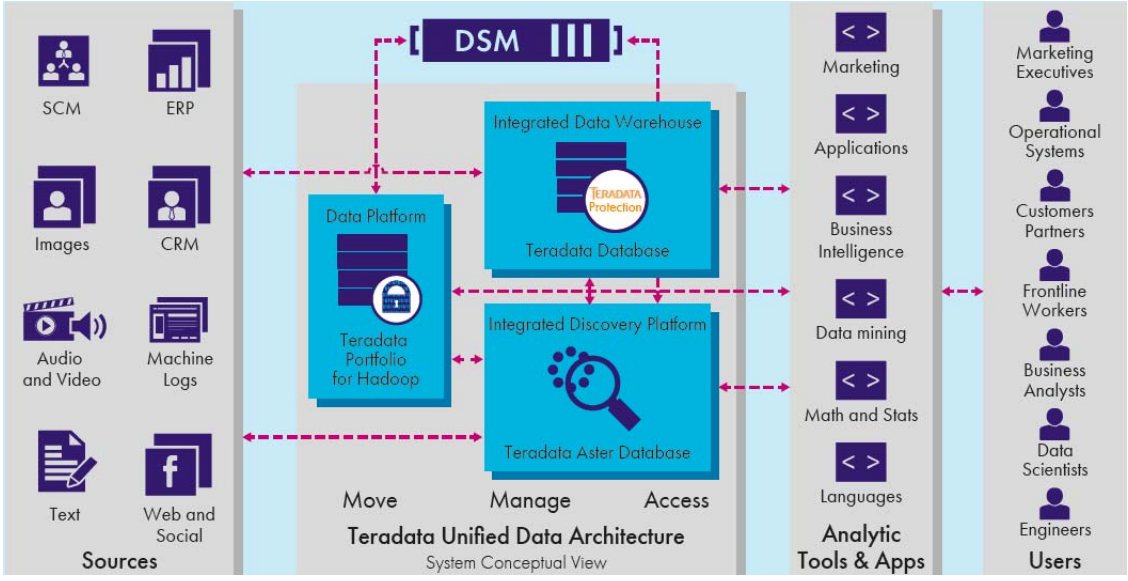
1 Vormetric Protection for Teradata Database (VPTD)

Vormetric Protection for Teradata Database (VPTD) enables column-level encryption of Teradata databases. This chapter describes how VPTD provides protection in the Teradata database environment.

Overview

A Teradata database consists of a Parsing Engine (PE) and any number of Access Module Processors (AMP) that exist on *nodes.* Nodes are servers that can host multiple virtual AMPs (VAMPs) and, optionally, the parsing engine itself.

Figure 2: Typical Teradata deployment



A Teradata site consists of one or more nodes. Each Teradata site requires a Key Manager -- either a Vormetric Data Security Manager (DSM) (as shown in Fugure 2) or a SafeNet KeySecure applicance.

The following Teradata **User Defined Functions** (UDFs) must be provided by Vormetric and installed in the Teradata database:

- *encrypt_cbc()*– Encrypts a string provided in Unicode format
- *decrypt_cbc()*– Decrypts data that was encrypted using *encrypt_cbc()* and returns a string in Unicode format
- *encrypt_int()*– Encrypts a string provided as an INTEGER datatype
- *decrypt_int()*– Decrypts data that was encrypted using *encrypt_int()* and returns an integer value
- *encrypt_byteint()*– Encrypts a string provided as a BYTEINT datatype
- *decrypt_byteint()*– Decrypts data that was encrypted using *encrypt_byteint()* and returns a byteint value
- *encrypt_smallint()*– Encrypts a string provided as a SMALLINT datatype
- *decrypt_smallint()*– Decrypts data that was encrypted using *encrypt_smallint()* and returns a smallint value
- *encrypt_date()*– Encrypts a string provided as a DATE datatype
- *decrypt_date()*– Decrypts data that was encrypted using *encrypt_date()* and returns a date value
- *encrypt_time()*– Encrypts a string provided as a TIME datatype
- *decrypt_time()*– Decrypts data that was encrypted using *encrypt_time()* and returns a time value
- *encrypt_timestamp()*– Encrypts a string provided as a TIMESTAMP datatype
- *decrypt_timestamp()*– Decrypts data that was encrypted using *encrypt_timestamp()* and returns a timestamp value
- *encrypt_fpe()*– Performs format preserving encryption on a string provided in Unicode format
- *decrypt_fpe()*– Decrypts data that was encrypted using *encrypt_fpe()*
- *encrypt_fpe_int()*– Performs format preserving encryption on a string provided as an INTEGER datatype
- *decrypt_fpe_int()*– Decrypts data that was encrypted using *encrypt_fpe_int()*
- *encrypt_fpe_byteint()*– Performs format preserving encryption on a string provided as a BYTEINT datatype
- *decrypt_fpe_byteint()*– Decrypts data that was encrypted using *encrypt_fpe_byteint()*
- *encrypt_fpe_smallint()*– Performs format preserving encryption on a string provided as a SMALLINT datatype

- *decrypt_fpe_smallint()*– Decrypts data that was encrypted using *encrypt_fpe_smallint()*
- *encrypt_ff1()*– Performs format preserving FF1 encryption on a string provided in Unicode format
- *decrypt_ff1()*– Decrypts data that was encrypted using *encrypt_ff1()*
- *encrypt_ff1_int()*– Performs format preserving FF1 encryption on a string provided as an INTEGER datatype
- *decrypt_ff1_int()*– Decrypts data that was encrypted using *encrypt_ff1_int()*
- *encrypt_ff1_byteint()*– Performs format preserving FF1 encryption on a string provided as a BYTEINT datatype
- *decrypt_ff1_byteint()*– Decrypts data that was encrypted using *encrypt_ff1_byteint()*
- *encrypt_ff1_smallint()*– Performs format preserving FF1 encryption on a string provided as SMALLINT datatype
- *decrypt_ff1_smallint()*– Decrypts data that was encrypted using *encrypt_ff1_smallint()*
- *encrypt_string()*– Provided for backward compatibility. Encrypts a string provided in Latin characters
- *decrypt_data()*– Provided for backward compatibility. Decrypts data that was encrypted using *encrypt_string()* and returns a string in Latin characters
- *encrypt_char()*– Encrypts Latin character sets. Note that this is not compatible with the ciphertext output of *encrypt_string()* for the given plain text and key combination.
- *decrypt_char()*– Decrypts Latin character sets.

Each Teradata node requires a connection to the Vormetric Data Security Manager (DSM) or SafeNet KeySecure applicance.

VPTD Operation

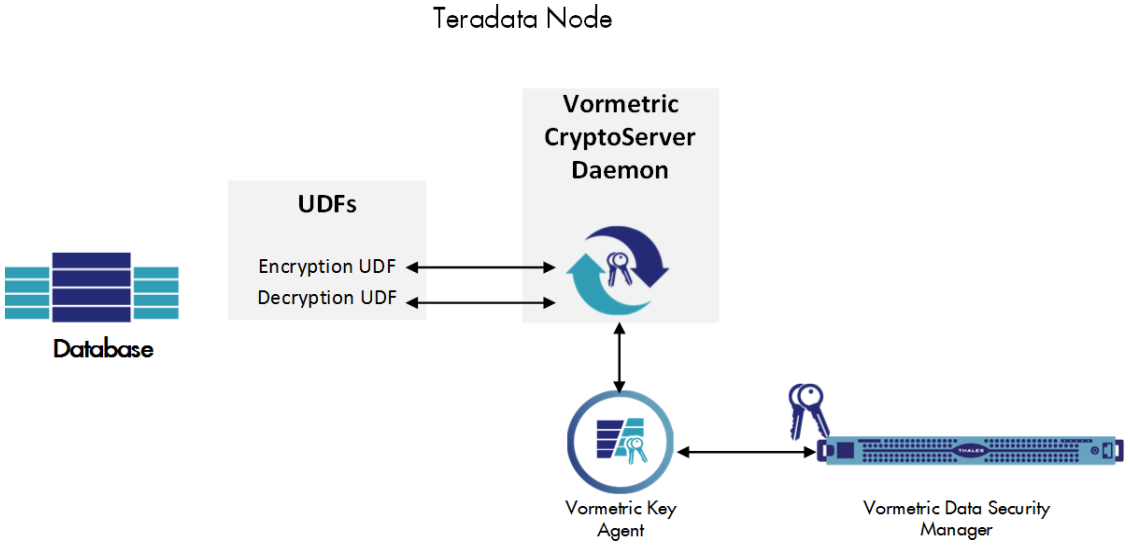
Once these components are installed, you have access to the UDFs that can be used to encrypt and decrypt Teradata database columns.[1] These UDFs can run in two modes: **Normal** Mode and **Fast** Mode.

Normal Mode

The following figure shows VPTD running in the Normal Mode.

1.You also have access to the VAE API library when configured with a Vormetric DSM. See the *VAE Installation Guide and API Reference* for details.

Figure 3: Teradata node with VPTD running in the Normal Mode with DSM



In the Normal mode, *encrypt_cbc()*, *encrypt_fpe()*, *encrypt_ff1()*, or *encrypt_string()* sends a cleartext string to the Vormetric Local Cryptoserver Daemon. The Key Agent obtains a key from the DSM and encrypts the cleartext string. The encrypted string is then returned.

Likewise, *decrypt_cbc()*, *decrypt_fpe()*, *decrypt_ff1()*, or *decrypt_data()* sends an encrypted text string to the Vormetric Local Cryptoserver Daemon. The Key Agent obtains the key from the DSM and decrypts the string. The cleartext string is then returned.

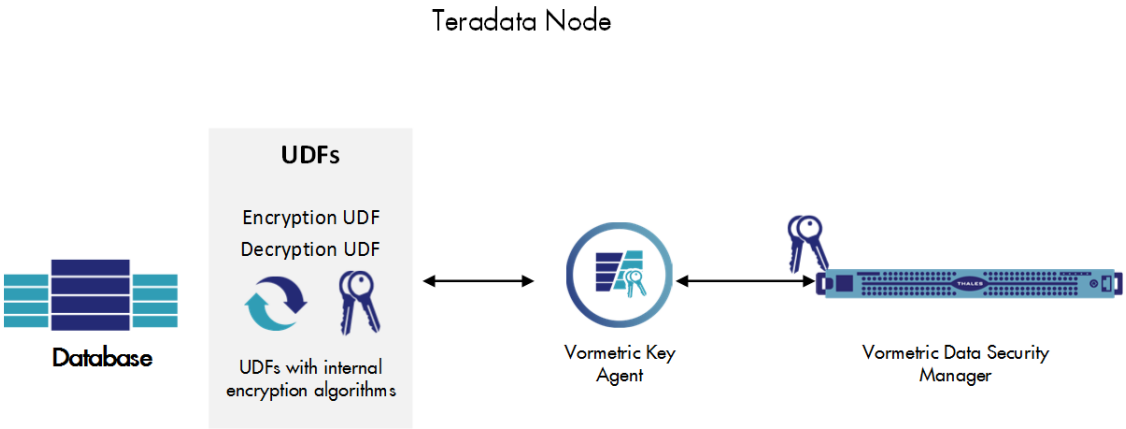
For more information about performance details and how to enable Normal and Fast Mode, see "Configure Vormetric Protection for Teradata" on page 27.

See "VPTD UDFs" on page 41 for examples of how to use these UDFs.

Fast Mode

The following figure shows VPTD running in the Fast Mode.

Figure 4: Teradata node with VPTD running in the Fast Mode



In the Fast Mode, the encryption UDF *encrypt_cbc()*, *encrypt_fpe()*, *encrypt_ff1()*, or *encrypt_string()* encrypts a cleartext string locally inside the UDF. The encrypted string is then returned.

Likewise, the decryption UDF *decrypt_cbc()*, *decrypt_fpe()*, *decrypt_ff1()*, or *decrypt_data()* decrypts an encrypted text string locally inside the UDF. The cleartext string is then returned.

For more information about performance details and how to enable Normal and Fast Mode, see "Configure Vormetric Protection for Teradata" on page 27.

See "VPTD UDFs" on page 41 for examples of how to use these UDFs.

Managing User Access Control (Whitelist and Blacklist)

VPTD supports coarse level access control and identity-based access control. Whitelisting and blacklisting VPTD users is particularly useful for preventing internal threats to your Teradata database.

To control which users can execute the UDFs, create blacklists and whitelists for the UDFs, add users to the *deny_decrypt.conf* black list and to prevent the user from executing decryption UDFs. Add users to the *allow_decrypt.conf* white list, to allow only that specific user to execute decryption UDFs.

By whitelisting approved users of the middleware (the business logic), all others, including the Teradata DB administrators, are blocked from accessing sensitive data.

For configuration options and details see "Configuring Access Control" on page 21.

Creating Profiles for Invoking UDFs

To streamline the invocation of the *encrypt_cbc()*, *decrypt_cbc()*, *encrypt_fpe()*, *decrypt_fpe()*, *encrypt_ff1()* and *decrypt_ff1()* UDFs, VPTD uses the file *profiles.conf*. This file contains named profiles that include several values required as input to these UDFs. Provide the profile name as a parameter when invoking the UDF. The UDF looks up the profile name and uses the parameters that are grouped under that profile name.

NOTE: Profiles are not supported for the *encrypt_string()* and *decrypt_data()* UDFs.

For more information, see “encrypt_char()” on page 65.

Components

This section describes the components of the Vormetric Data Security (VDS) product that are relevant to Application Encryption.

Key Manager: Data Security Manager or KeySecure

Use a key manager as a policy engine and a central key and policy manager. VPTD supports either the Vormetric Data Security Manager (DSM) or Gemalto SafeNet KeySecure. The key manager stores and manages host encryption keys, data access policies, administrative domains, and administrator profiles.

Key Agent

The Vormetric Key Agent provides a library that implements the PKCS#11 interface. This library is a dynamically loadable library (dll) on Windows and a shared object (so) on Linux and UNIX. The Key Agent's PKCS#11 library communicates over a secure channel to the key manager for all significant functionality. This is sometimes called the *Vormetric Application Encryption Agent*.

Vormetric Local Cryptoserver Daemon

In the Normal mode, Vormetric Local Cryptoserver Daemon accepts requests from the UDFs, forwards them to the Vormetric Key Agent for processing, and returns the Key Agent output to the respective UDF.

2

Installing and Deploying VPTD

This chapter describes how to install and configure the Vormetric Protection for Teradata Database on your Teradata host. It contains the following sections:

Prerequisites for Installing VPTD on a Teradata Node

The following is required to install Vormetric Protection for Teradata Database:

- A hardware key manager, either Vormetric DSM or SafeNet KeySecure. To find your DSM version, open the Management Console and choose **System > About**). For DSM installation instructions, see *DSM Installation and Configuration Guide*. For KeySecure installation instructions, refer to your KeySecure documentation.
- Access to the DSM or KeySecure key manager from all Teradata nodes.
- If using a Vormetric DSM key manager, refer to the *Vormetric Data Security Platform DSM | VAE | VTS| VKM | VPTD | BDT Compatibility Matrix* for complete details on supported configurations of Teradata versions and operating systems with VPTD versions. If using a SafeNet KeySecure key manager, refer to the KeySecure release notes for similar supported configurations.

  For each node you need:

  - Root password to Teradata nodes

- The IP address or FQDN of the node
- Installation executable file for Vormetric Protection for Teradata version 6.4.0 for Teradata. For example:

  `vptd16.10-6.4.0-47-sles11sp1-x86_64.bin`

  This installs the following:
  - Vormetric Protection for Teradata Database, including the Vormetric Key Agent software
  - UDFs
  - Vormetric Local Cryptoserver Daemon
  - Sample installation script (`install_udfs.bteq.sample`)
  - Sample UDF profile configuration file (`profiles.conf.sample`)
  - Sample Cryptoserver configuration file (`vormetric_local_crypto_server.conf.sample`)
  - Mask configuration file (`mask.conf.sample`)
  - ICAPI configuration files: `icapi.conf` and `ProtectAppICAPI.properties`

Installation Overview

The following are the high-level steps for installing Vormetric Protection for Teradata Database on a Teradata node.

1. Collect configuration information in the "Vormetric Protection for Teradata Installation Checklist" on page 13.
2. "Add the Teradata Node to the DSM Database" on page 17.
3. "Install Vormetric Protection for Teradata on the Teradata Node" on page 18.
4. "Configure Vormetric Protection for Teradata" on page 27.
5. "Deploy Vormetric Protection on a Teradata Node" on page 34.

Vormetric Protection for Teradata Installation Checklist

Use this table to verify prerequisites and collect the information you need for the installation.

Table 1: VPTD Installation Checklist

	Status
Hardware DSM, version 6.1.0 or later, or SafeNet KeySecure	
Obtain Vormetric Protection for Teradata installation file from Vormetric. Example: (vptd16.10-6.4.0-47-sles11sp1-x86_64.bin)	
Host system clock set to the correct time zone.	
Fully Qualified Domain Name (FQDN) of the DSM or KeySecure key manager (page 13)	
Root password for the Teradata node	
IP address or Fully Qualified Domain Name (FQDN) of the Teradata node. (page 14)	
Symmetric encryption key, cached on the host. (page 15)	

How to Get the FQDN of the Key Manager (DSM or KeySecure)

DSM FQDS

Log in to the Management Console and look at the dashboard.

1. Open a browser and enter the DSM URL.

   This is either the host name if configured in DNS, or its IP over HTTPS of the DSM.

2. Enter the default login and password. The default login is `admin`. The default password is `admin123`.

NOTE: You are asked to change the default password upon first login. Remember this new password or you will not be able to log in again!

3. The FQDN is at the top.

KeySecure FDQN

1. Connect to your KeySecure instance:

```
ssh -i {key file} ksadmin@ip.address
```

1. Use the hostname command to view the hostname:

```
ksadmin@keysecure:~$ hostname
```

Teradata Node Name Resolution

You can map a Teradata node name to an IP address using a Domain Name Server (DNS). DNS is the most preferred method of host name resolution.

You can also modify the `hosts` file on the DSM or identify a host using only the IP address.

- If you use DNS to resolve host names, use the FQDN for the host names.
- If you do NOT use a DNS server to resolve host names, do the following on all of the DSMs and the protected Teradata nodes:
  - **Modify the `hosts` file on the DSM:** To use names like *serverx.domain.com,* enter the node names and matching IP addresses in the `/etc/hosts` file on the DSM using the `host` command under the `network` menu of the Admin CLI (see See “Using the Admin CLI” on page 15. For example:

```
network$ host add grimes-dev1 192.168.42.12
SUCCESS: add host
network$ host sh
name=localhost6.localdomain6 ip=::1
name=linux32-48215.sacdbackup.com ip=192.168.48.215
name=rgrimes-dev1 ip=192.168.42.12
SUCCESS: show host
```

    You must do this on *each* DSM, since entries in the host file are not replicated across DSMs.
  - **Modify the `hosts` file on the protected hosts:** Enter the DSM host names and matching IP addresses in the `/etc/hosts` file on the Teradata node. *You must do this on EACH protected node, making sure to add an entry for all DSM nodes (if using HA).*

OR

- **Use IP addresses:** If using IP addresses as protected Teradata node names, you must enable Agent IP in the DSM (see “agentip” Enable/Disable in the “network” menu in the CLI. This is described in “Using the Admin CLI” on page 15). With Agent IP enabled, you can have some hosts identified by host name and some by IP simultaneously. In other words, they don't all need to use an IP address when Agent IP is enabled.

Using the Admin CLI

Access the CLI menu as follows:

1. Start the serial console application.
2. If the login prompt is not displayed, press the **Enter** key to wake up the connection.
3. Log into the appliance. The default System Administrator name and password are `cliadmin` and `cliadmin123`.

Example

At the prompt, type `cliadmin` followed by the password. At the prompt type `network`. See the example below:

```
network$ agentip show
agent ip address support : off
SUCCESS: agent ip address support showed.
network$ agentip on
WARNING: The Security Server will restart automatically after enabling
agent IP address support!
Continue? (yes|no)[no]:yes
SUCCESS: Agent IP address support is enabled and the server restarted.
network$ agentip show
agent ip address support : on
SUCCESS: Agent IP address support showed.
network
```

Obtaining a Data Encryption Key for Your VPTD Deployment

This section contains two subsections for DSM users:

- "Creating a Data Encryption Key" on page 15
- "Using an Encryption Key Not Created with DSM5.2.2or later" on page 17

NOTE: If you use a KeySecure key manager, see the KeySecure Administration Guide for details on how to create a data encryption key suitable for VPTD.

Creating a Data Encryption Key

1. Go to **Keys > Agent Keys > Keys** in the Management Console to open the *Agent Keys* window.
2. Click **Add** to open the **Add Agent Key** window.
3. Enter a key name, description, and security algorithm.
   - **Name**: Name of key. 64 character limit.
   - **Description**: Optional key description. 265 character limit.

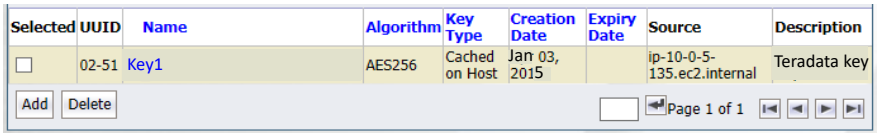
- **Template**: A key template with a set of predefined attributes. To create a valid Teradata key, select **Default_SQL_Symmetric_Key_Template** and do not change any of the custom attribute values.
- **Algorithm**: Algorithm used to create the key.
- **Key Type**: Location for the encryption key. **Stored on Server** keys are downloaded to non-persistent memory on the host. Each time the key is needed, the host must retrieve the key from the DSM. **Cached on Host** downloads and stores (in an encrypted form) the key in persistent memory on the host. For performance reasons, **Cached on Host** is highly recommended for Teradata installations. For Fast Mode (configuration file setting '`udfaes` on', see "Deploy Vormetric Protection on a Teradata Node" on page 34), key type **Cached on Host** is MANDATORY.
- **Unique to Host**: This check box is displayed when **Cached on Host** is selected. When enabled, it makes the encryption key unique. The key is downloaded to the host, encrypted using the host password, and stored. These keys are used for locally attached devices, as files encrypted by them can be read only by one machine. Do not enable this check box for cloned systems, RAID configurations, clustered environments, or any environment that uses host mirroring. Requires that **Key Creation Method** is set to *Generate.*
- **Key Creation Method**: Select to generate a key using a random seed (**Generate**) or by **Manual Input**.
- **Expiry Date**: Date the key expires.
- **Key Refreshing Period (minutes)**: Used only with Oracle Database TDE and Microsoft SQL Server TDE. How long to keep the key in the local key cache before it is refreshed.

  Example:

  Name: Key1
  Description: Teradata key
  Algorithm:AES256

  All other values are the default.

4. Click **OK**. Your new key is created and displayed in the *Agent Keys* window.



5. Create as many keys as desired.

Using an Encryption Key Not Created with DSM5.2.2or later

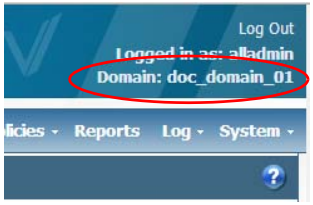
If you try to use a key created with DSM 5.2.1 or another environment in the VPTD environment, it will not work. You must first modify the key's attributes in the DSM Management Console GUI (**Keys** > [Select a key] > **Attributes):**

Cryptographic Usage Mask	127
Object Type	Symmetric Key
x-VormApplicationName	Vormetric Key Agent
x-VormCanBePlainText	true
x-VormCanNeverBeExported	true
x-VormCanNeverBePlaintext	true
x-VormCanObjectPersist	true

Add the Teradata Node to the DSM Database

Your Teradata node names must be added to the DSM database before Vormetric Protection for Teradata is installed. This section describes how to do this. To add the Teradata node to the DSM database, you must have the Teradata node's Fully Qualified Domain Name (FQDN—54 character max) or IP address.

1. Log on to the Management Console as a Security Administrator with *Key* and *Policy* roles or as an administrator of type *All*.
2. Switch to the domain containing the Teradata node you want to protect. Click **Domains > Switch Domains.** The *Switch Domains* window opens.
3. Select the domain that will contain the Teradata host and click **Switch to domain**. The domain in which you are working is displayed in the upper right corner of the Management Console.



4. Select **Hosts > Hosts** in the menu bar. An empty *Hosts* window opens.
6. Click **Add**. The *Add Host* window opens.
7. Enter the following information:
    - **Host Name**: Enter FQDN of the Teradata Node. The IP address or host name will work, but the FQDN changes the least. Host names cannot contain an underscore.

- **Description**: Optional. Enter text to identify the node or its function. Limited to 256 characters.
- **Registration Allowed Agents**: Select the agents that will run on the node system. In this case it will be **Key**.
- **License Type**: Choose the type of license that will run on this host. Options are **Perpetual**, **Term**, and **Hourly**, depending on the system license.

8. Click **Ok**. You are returned to the *Hosts* window.
9. Click the host name link that you just added to the DSM database. This opens the **General** tab of the *Edit Host* window. Make sure the **Registration Allowed** and **Communication Enabled** check box are checked for the Key Agent.
10. Your node is added to the DSM database.
11. Repeat for all your protected Teradata nodes.

Configure NAE interface mode on KeySecure

If using a KeySecure key manager, configure the NAE interface mode on the KeySecure server:

1. Log in to KeySecure.
2. Go to **Admin Setting > System > Interfaces**.
3. Select the appropriate NAE interface mode:
   - For TCP communication - Select any **No TLS** mode.
   - For SSL communication - Select any mode except **No TLS**.

Install Vormetric Protection for Teradata on the Teradata Node

The following steps describe how to install Vormetric Protection for Teradata for the first time.

Install Vormetric Protection for Teradata

1. Log on with root access to the host where you will install Vormetric Protection for Teradata.
2. Copy or mount the installation file to the host system.
3. Start the installation. Type:

```
# ./vptd-<product-version-build-system>.bin
```

Example:

```
vptd16.10-6.3.0-47-sles11sp1-x86_64.bin
```

4. The text of the License Agreement appears. Page through the agreement, then type y and press **Enter** to accept. The installation proceeds.
5. At the agent registration prompt, type y or press **Enter** to proceed with registration, or type n if you plan to register later.

```
Do you want to continue with agent registration? (Y/N) [Y]:
```

6. Continue to follow the prompts:

   a: Enter the fully qualified host name of the primary DSM or KeySecure appliance, and then press **Enter.**

   b: Verify the host name, and then press **Enter**.

NOTE: The following Steps 6c through 11 are applicable only when the nameof a DSM host is entered. These Steps 6c through 11 are *not* applicable if you enter the host name of a KeySecure server.

   c: The installer shows a numbered list of host names and prompts you to choose one. From the list, type the number of the host name of your local machine, and then press Enter. This name must match the name of the host that was added to the DSM or KeySecure applicance by the Security Administrator.

   Example:

```
Please enter the host name of this machine, or select from
the following list. The name you provide must precisely
match the name used on the “Add Host” page of the
Management Console.
[1] host1.example.com
[2] sys41017-priv.example.com
[3] sys41017-vip.example.com
[4] 10.3.41.127
Enter a number, or type a different host name or IP
address in manually:
What is the name of this machine? [1]: 1
Generating certificate...done.
Signing certificate...done.
```

7. At the following prompt, choose whether you want to register to the Security Server using a shared secret or using fingerprints.

```
Would you like to register to the shared Security Server using a
registration shared secret (S) or using fingerprints (F)? (S/F)
[S]:
```

8. To enable FF1 licensing, enter **Y** at the following prompt:

```
Do you want to enable FF1 License for this host? (Y/N) [N]:
```

   The rest of this procedure is based on fingerprint registration. The advantages and disadvantages of each security method are beyond the scope of this document. Refer to the DSM documentation for complete details.

9. At the prompt, press **Enter** to enable cloning prevention ability, by associating the installation with existing hardware.

   Example:

```
It is possible to associate this installation with the hardware
of this machine. If selected, the agent will not contact the DSM
or use any cryptographic keys if any of this machine's hardware is
changed. This means that if the machine is copied (for example, a
clone of a virtual machine), the copy will not function
correctly. This can be rectified by running this registration
program again.
Do you want to enable this functionality?  (Y/N) [Y]:
```

10. The CA certificate fingerprint is displayed:

```
The following is the fingerprint of the CA certificate. Please
verify that it matches the fingerprint shown on the Dashboard page
of the Management Console.  If they do not match, it can indicate
an unsuccessful setup or an attack.
4C:10:12:72:1B:56:7D:93:A4:71:0D:93:B9:47:CF:A0:8F:D4:5A:E9
Do the fingerprints match? (Y/N) [N]: Y
```

    At this stage of the installation, you, the host administrator, and DSM Security Administrator must exchange information to confirm that the agent host and DSM share valid certificates. This step verifies that nobody is intercepting and modifying traffic between the DSM and agent. It is a security feature.

    a. **Host Admin:** Send the fingerprint to the DSM Security Administrator and wait for confirmation.

    b. **DSM Security Admin**:

- Log on to the DSM Management Console and navigate to the domain where the Teradata node was added.
- Click the **Dashboard** tab.
- Match the fingerprint from the Host Admin with the **EC CA fingerprint** on the Dashboard.
- Advise the Host Admin of the results.

c. **Host Admin:** If the fingerprints match, type y and then press **Enter**. “Installation success.” is displayed.

NOTE: If this is a Key Agent installation, you are prompted for a password for the Key Agent library.

11. Enter the password for the Key Agent library and confirm your entry. Password minimum length for PIN is 8 characters and maximum length is 63 characters.

```
Please enter a password for the Key Agent library.
Accesses to this library will be protected by this password.
NOTE: If using Oracle RAC, passwords must be the same on all nodes.
Please enter password:
Enter again to confirm:
Successfully registered the Vormetric Key Agent with the primary
Vormetric Data Security Server on test-vormetric.com.
```

NOTE: Do not forget this password. You will need it later during Vormetric Protection for Teradata Database deployment. To change the password, you must re-register the agent.

Configuring Access Control

VPTD provides the following user access control configuration options:

- “Identity-Based Access Control (Recommended)” on page 22
- “Coarse-Level Access Control (Default)” on page 24

Access control parameters are defined in the `vormetric_local_cryptoserver.conf` file. Set `identityacl` to on to indicate identity-based access control is enforced.

When `identityacl` is set to off, or not configured, coarse-level access control is enforced by default. Identity-based access control uses the same files (`allow_encrypt.conf` and `allow_decrypt.conf`) to contain the access control definitions.

NOTE: The formats of coarse-level and identity-based access are different in the whitelist (`allow_encrypt.conf` and `allow_decrypt.conf`)files. However, the format of the blacklist (`deny_encrypt.conf` and `deny_decrypt.conf`) files are same for both modes of access control.

Identity-Based Access Control (Recommended)

Use the identity-based access control to control access at the database column level. Keys can be assigned to encrypt/decrypt one or more database columns, which provides column-level access control based on identity.

Access to database encryption and decryption operations are defined in the `allow_encrypt.conf` and `allow_decrypt.conf` configuration files, based on key-value pairs. The encryption key is the key, and a comma-separated list of users who may access that encryption key is the associated value.

Modified Whitelist for Decryption

The modified `allow_decrypt.conf` file has the following format:

```
[<denied behavior>]
<key name>:<comma separated list of allowed users>
```

Where *`denied behavior`* is one of the following:

- **deny -** Access is denied. This is the default behavior
- **blank** - Blank value will be returned.
- **ciphertext** - Encrypted column value will be returned.
- **zero** - Zero (0) will be returned
- **“*custom text*”** - An administrator-defined custom string specified under double quotes, for example: “`Access Denied`”.

NOTE: The **blank** and ***custom text*** denied behaviors are not supported in UDFs that have an integral return type

A sample `allow_decrypt.conf` file:

```
[ciphertext]
Name_Address_key:User1,User3

[“Access Denied”]
Phone_key:User2,User3

[zero]
SSN_key:User3
```

NOTE: Multiple denied behaviors for the same key is not allowed in identity-based access control. Do not specify the same key under multiple denied behavior entries, otherwise an error occurs at the BTEQ prompt.

More Configuration Notes

- Multiple key-user mappings can share the same denied behavior.
- The wildcard asterisk (*****) denotes all users. When ***** is used, the denied behavior field is ignored.

NOTE: For installation specifics see step 2 of “Deploy Vormetric Protection on a Teradata Node” on page 34.

Modified Whitelist for Encryption

The modified `allow_encrypt.conf` has the following format:

```
<key name> : <comma separated list of allowed users>
```

Consider the following example of a `allow_encrypt.conf` configuration:

```
Name_Address_key:User1,User3
Phone_key:User2,User3
SSN_key:User3
```

NOTE: Unlike decryption, encryption does not support any custom deny behavior. When a user attempts to encrypt a column using a key that the user does not have access to, the operation is denied.

Coarse-Level Access Control (Default)

With coarse-level access control, users listed in the white list can access all database tables.

NOTE: Identity-based access control goes one step further, by specifying which key can be used by which user.

To configure coarse-level control, decide whether you are *blacklist-centric*—you have a list of users to block access to a command and all others are unblocked, or *whitelist-centric*—you have a list of users to allow access to a command and all others are blocked. Choose which orientation makes the most sense for your situation, then create the following four files in `/etc/vormetric` with 600 permissions (read/write by root only):

```
allow_encrypt.conf
allow_decrypt.conf
deny_encrypt.conf
deny_decrypt.conf
```

Add the logins of users, one line per user, to either deny (blacklist) or allow (whitelist) access to do encrypting or decrypting. User names must be entirely capitalized. The wildcard character (*) indicates “all users” and it is allowed in the whitelist, but not the blacklist. No other regular expressions are permitted. Make your configuration blacklist-centric by putting just a * in the whitelist and adding user names in the blacklist. Make it whitelist-centric by leaving your blacklist empty and putting user names in the whitelist.

If you are whitelist-centric, only users in `allow_decrypt.conf` will be able to decrypt data, and only users in `allow_encrypt.conf` will be able to encrypt data. If you are blacklist-centric, only users in `deny_decrypt.conf` will not be able to decrypt data, and only users in `deny_encrypt.conf` will be not able to encrypt data.

We recommend whitelisting approved users of the middleware (the business logic). This will block all others, including the Teradata DB administrators from accessing sensitive data.

The Linux system administrator must then replicate these files to all other Teradata nodes.

Verify the Installation

The following steps verify installation.

1. Run the following commands on the Teradata node to confirm that all components of VPTD were entered into the rpm database, which means that VPTD was installed successfully.

   ```
   rpm -qa | grep vae-td
   rpm -qa | grep vee-key
   ```

2. Open the DSM Edit Host window and verify that **Registration Allowed** and **Communication Enabled** are selected for the Key Agent.

Modify the Key Cache

You can modify two settings in the key cache at application level or on the DSM.

By default, keys are cached on the host, with a time-to-live value of 10080 minutes (7 days).

- You can choose to cache the key on the DSM or on the host.
- You can also modify how long the key stays in the local key cache before it is re-fetched from the DSM.

By accepting the default setting “Cached on Host”, the cryptographic key will be cached within the Key Agent library on the particular host (Teradata node).

The setting “Stored on Server” means that the key never leaves the DSM and thus all cryptographic operations with this key are performed on the DSM itself, not on the host. This setting is strongly discouraged.

To modify the key cache on the DSM:

1. Log on to the DSM as an administrator of type Security Administrator.
2. Click the **Domains** tab, and switch to the domain where the key is installed.
3. Click **Keys > Agent Keys**, and then click on the name of the key you want to modify. If you have many keys, search for a key on the Management Console using the **Keyname contains**: field. The *Edit Agent Key* window opens.
4. In the **Key Type** drop-down menu, select **Cached on Host** or **Stored on Server** (DSM).
5. In the **Key Refreshing Period (minutes)** field, enter the number of minutes you want the key in the local key cache before it is refreshed from the DSM.
6. Click **Ok**.

Install VPTD in Silent Mode

A Silent Mode installation means that all requisite input is provided in a file and VPTD is invoked using the `-s` option. To install VPTD in Silent Mode, follow the instructions for the key manager you are using:

- “Silent Mode Install with DSM” on page 26
- “Silent Mode Install with KeySecure” on page 27

Silent Mode Install with DSM

1. Create a config file containing following fields:

```
SERVER_HOSTNAME=<hostname>
AGENT_USEIP=Yes
PKCS11_PASSWORD=<password>
```

2. Obtain VPTD and copy it to the directory that contains the config file that you just created.
3. Enter:

```
# ./vptd<product-version-build-system>.bin -s <config file path>
```

Example

```
# ./vptd16.10-6.4.0-47-sles11sp1-x86_64.bin -s <config file path>
```

The following fields are optional for the config file:

Table 2: Options for the Silent Mode installation with DSM configuration file

Name	Function	Required
SERVER_HOSTNAME	Host name ofKeySecure key manager	Yes
SERVER_IP	Synonym for SERVER_HOSTNAME	No
AGENT_USEIP	Uses IP address instead of host name	No
AGENT_HOST_NAME	Uses the name of the machine instead of IP	No
AGENT_HOST_PORT	Port number to expose	No
STRONG_ENTROPY	Use `/dev/random` on linux	No
PKCS11_STANDALONE	Non-server mode for PKCS11 agent	No
PKCS11_PASSWORD	Password for PKCS11 agent	Yes (`pkcs11 only`)
ONEWAY_COMMS	Set when one-way communication required	No
USEHWSIG	Associate hardware to keys+certs (1|0)	No
SHARED_SECRET	Use given registration shared secret	No

Table 2: Options for the Silent Mode installation with DSM configuration file

Name	Function	Required
HOST_DOMAIN	Registration domain	Yes (If SS provided)
HOST_GROUP	Registration host group	No
HOST_DESC	Host description	No
FF1_ENABLED	Enable use of FF1 license	No

Silent Mode Install with KeySecure

1. Create a config file containing following fields:

```
SERVER_HOSTNAME=<hostname>
```

2. Obtain VPTD and copy it to the directory that contains the config file that you just created.
3. Enter:

```
# ./vptd<product-version-build-system>.bin -s <config file path>
```

Example

```
# ./vptd16.10-6.4.0-47-sles11sp1-x86_64.bin -s <config file path>
```

The following field is mandatoryfor the config file:

Table 3: Silent Mode with KeySecure installation configuration file

	Function	Required
SERVER_HOSTNAME	Host name of the KeySecure key manager	Yes

Configure Vormetric Protection for Teradata

After installing VPTD as described in "Install Vormetric Protection for Teradata on the Teradata Node" on page 18, do the following steps to configure VPTD and get the system up and running.

NOTE: This procedure must be done only once regardless of the number of nodes.

1. Choose between Normal Mode and Fast Mode.

   Before configuring Vormetric Protection for Teradata, you must first determine whether you will run in the Normal Mode or the Fast Mode. This decision will affect the configuration settings.

   The table below outlines the various attributes of each mode and how they are enabled.

Table 4: Comparison of Normal Mode and Fast Mode

	Normal Mode (`udfaes off`)	**Fast Mode** (`udfaes on`)
Teradata Protected Mode (comment out the `alter function` lines in install_udfs.bteq)	• Supported by 5.2.2 and later releases. • Encryption/decryption done on cryptoserver.	• 5.2.3 and later releases. • Encryption/decryption done locally inside the respective UDF.
Teradata Unprotected Mode (Default)	Not generally used.	• Recommended for maximum performance. • Encryption/decryption done locally inside the respective UDF.

   Specify the Teradata Protected or Unprotected Mode in the BTEQ file. Specify the `udfaes` parameter (Fast Mode or Normal Mode) in the Vormetric Local Cryptoserver Daemon configuration file (see "Deploy Vormetric Protection on a Teradata Node" on page 34).

2. Create a Basic Teradata Query (BTEQ) script from the provided sample to install the UDFs.

   a. Change directories:

   ```
   # cd /opt/vormetric/DataSecurityExpert/agent/pkcs11/teradata/udfs/
   ```

   b. Copy the UDF sample script `install_udfs.bteq.sample` to a file named `install_udfs.bteq` in the same directory. Keep `install_udfs.bteq.sample` as a reference, and use `install_udfs.bteq` as your working BTEQ script.

Edit `install_udfs.bteq` as per the embedded instructions. Replace the words in capital letters with real values. For example, change USERNAME and PASSWORD to a real user name and password.

```
bteq << $EOF

* Replace Teradata USERNAME and PASSWORD with a site-specific username
and password.

.logon USERNAME,PASSWORD;

* Replace DBC with the database from which you want to derive the
vormetric user.
* Note that the USER who installs the UDFs into the system must be set to
the latin
* char set. After the UDFs are installed, the character set for this
particular
* user MAY be changed to something else, for instance UNICODE. But during
the UDF
* installation, the user's character set MUST be latin.

create user vormetric from DBC as perm=10000000 password=SOMEPASSWORD
default character set latin;

grant create function on vormetric to vormetric;
grant alter function on vormetric to vormetric;
grant drop function on vormetric to vormetric;
grant execute function on vormetric to public;

.logoff
.logon vormetric,SOMEPASSWORD;
```

```
replace function encrypt_string (inputString varchar(16384), inputKeyname
varchar(256)) returns varbyte(16000) specific encrypt_string language c no sql
not deterministic parameter style sql called on null input external name
'co:udf_encrypt_string:./udf_encrypt_string.o';

replace function decrypt_data (inputString varbyte(16000), inputKeyname
varchar(256)) returns varchar(16384) specific decrypt_data language c no sql
not deterministic parameter style sql called on null input external name
'co:udf_decrypt_data:./udf_decrypt_data.o';

replace function encrypt_byteint (inputdata byteint, inputKeyname varchar(256))
returns varbyte(16000) specific encrypt_byteint language c no sql not
deterministic parameter style sql called on null input external name
'F:encrypt_byteint';

replace function decrypt_byteint (inputString varbyte(16000), inputKeyname
varchar(256)) returns byteint specific decrypt_byteint language c no sql not
deterministic parameter style sql called on null input external name
'F:decrypt_byteint';

replace function encrypt_smallint (inputdata smallint, inputKeyname
varchar(256)) returns varbyte(16000) specific encrypt_smallint language c no
sql not deterministic parameter style sql called on null input external name
'F:encrypt_smallint';

replace function decrypt_smallint (inputString varbyte(16000), inputKeyname
varchar(256)) returns smallint specific decrypt_smallint language c no sql not
deterministic parameter style sql called on null input external name
'F:decrypt_smallint';

replace function encrypt_int (inputString integer, inputKeyname varchar(256))
returns varbyte(16000) specific encrypt_int language c no sql not deterministic
parameter style sql called on null input external name 'F:encrypt_int';

replace function decrypt_int (inputString varbyte(16000), inputKeyname
varchar(256)) returns integer specific decrypt_int language c no sql not
deterministic parameter style sql called on null input external name
'F:decrypt_int';

replace function decrypt_ff1_smallint (inputString varchar(8192) CHARACTER SET
LATIN, inputKeyname varchar(256)) returns smallint specific
decrypt_ff1_smallint language c no sql not deterministic parameter style sql
called on null input external name 'F:decrypt_ff1_smallint';

replace function decrypt_ff1_int (inputString varchar(8192) CHARACTER SET
LATIN, inputKeyname varchar(256)) returns integer specific decrypt_ff1_int
language c no sql not deterministic parameter style sql called on null input
external name 'F:decrypt_ff1_int';
```

```
replace function encrypt_time (inputdata time, inputKeyname varchar(256)) returns
varbyte(16000) specific encrypt_time language c no sql not deterministic parameter
style sql called on null input external name 'F:encrypt_time';

replace function decrypt_time (inputString varbyte(16000), inputKeyname
varchar(256)) returns time specific decrypt_time language c no sql not
deterministic parameter style sql called on null input external name
'F:decrypt_time';

replace function encrypt_date (inputdata date, inputKeyname varchar(256)) returns
varbyte(16000) specific encrypt_date language c no sql not deterministic parameter
style sql called on null input external name 'F:encrypt_date';

replace function decrypt_date (inputString varbyte(16000), inputKeyname
varchar(256)) returns date specific decrypt_date language c no sql not
deterministic parameter style sql called on null input external name
'F:decrypt_date';

replace function encrypt_timestamp (inputdata timestamp, inputKeyname varchar(256))
returns varbyte(16000) specific encrypt_timestamp language c no sql not
deterministic parameter style sql called on null input external name
'F:encrypt_timestamp';

replace function decrypt_timestamp (inputString varbyte(16000), inputKeyname
varchar(256)) returns timestamp specific decrypt_timestamp language c no sql not
deterministic parameter style sql called on null input external name
'F:decrypt_timestamp';

replace function encrypt_char (inputString varchar(16384), inputKeyname
varchar(256), inputCharcolumnsize INTEGER) returns varbyte(16000) specific
encrypt_char language c no sql not deterministic parameter style sql called on null
input external name 'co:udf_encrypt_char:./udf_encrypt_char.o';

replace function decrypt_char (inputString varbyte(16000), inputKeyname
varchar(256), inputCharcolumnsize INTEGER) returns varchar(16384) specific
decrypt_char language c no sql not deterministic parameter style sql called on null
input external name 'co:udf_decrypt_char:./udf_decrypt_char.o';

replace function encrypt_cbc (inputString varchar(8192) CHARACTER SET UNICODE,
inputKeyname varchar(256)) returns varbyte(16000) specific encrypt_cbc language c
no sql not deterministic parameter style sql called on null input external name
'co:udf_encrypt_cbc:./udf_encrypt_cbc.o';

replace function decrypt_cbc (inputString varbyte(16000), inputKeyname
varchar(256)) returns varchar(8192) CHARACTER SET UNICODE specific decrypt_cbc
language c no sql not deterministic parameter style sql called on null input
external name 'co:udf_decrypt_cbc:./udf_decrypt_cbc.o';

replace function encrypt_fpe (inputString varchar(8192) CHARACTER SET UNICODE,
inputKeyname varchar(256)) returns varchar(8192) CHARACTER SET UNICODE specific
encrypt_fpe language c no sql not deterministic parameter style sql called on null
input external name 'co:udf_encrypt_fpe:./udf_encrypt_fpe.o';

replace function decrypt_fpe (inputString varchar(8192) CHARACTER SET UNICODE,
inputKeyname varchar(256)) returns varchar(8192) CHARACTER SET UNICODE specific
decrypt_fpe language c no sql not deterministic parameter style sql called on null
input external name 'co:udf_decrypt_fpe:./udf_decrypt_fpe.o';
```

```
replace function encrypt_fpe_byteint (inputString byteint, inputKeyname
varchar(256)) returns varchar(8192) CHARACTER SET LATIN specific
encrypt_fpe_byteint language c no sql not deterministic parameter style sql called
on null input external name 'F:encrypt_fpe_byteint';

replace function encrypt_fpe_smallint (inputString smallint, inputKeyname
varchar(256)) returns varchar(8192) CHARACTER SET LATIN specific
encrypt_fpe_smallint language c no sql not deterministic parameter style sql called
on null input external name 'F:encrypt_fpe_smallint';

replace function encrypt_fpe_int (inputString integer, inputKeyname varchar(256))
returns varchar(8192) CHARACTER SET LATIN specific encrypt_fpe_int language c no
sql not deterministic parameter style sql called on null input external name
'F:encrypt_fpe_int';

replace function decrypt_fpe_byteint (inputString varchar(8192) CHARACTER SET
LATIN, inputKeyname varchar(256)) returns byteint specific decrypt_fpe_byteint
language c no sql not deterministic parameter style sql called on null input
external name 'F:decrypt_fpe_byteint';

replace function decrypt_fpe_smallint (inputString varchar(8192) CHARACTER SET
LATIN, inputKeyname varchar(256)) returns smallint specific decrypt_fpe_smallint
language c no sql not deterministic parameter style sql called on null input
external name 'F:decrypt_fpe_smallint';

replace function decrypt_fpe_int (inputString varchar(8192) CHARACTER SET LATIN,
inputKeyname varchar(256)) returns integer specific decrypt_fpe_int language c no
sql not deterministic parameter style sql called on null input external name
'F:decrypt_fpe_int';

replace function encrypt_ff1 (inputString varchar(8192) CHARACTER SET UNICODE,
inputKeyname varchar(256)) returns varchar(8192) CHARACTER SET UNICODE specific
encrypt_ff1 language c no sql not deterministic parameter style sql called on null
input external name 'co:udf_encrypt_ff1:./udf_encrypt_ff1.o';

replace function decrypt_ff1 (inputString varchar(8192) CHARACTER SET UNICODE,
inputKeyname varchar(256)) returns varchar(8192) CHARACTER SET UNICODE specific
decrypt_ff1 language c no sql not deterministic parameter style sql called on null
input external name 'co:udf_decrypt_ff1:./udf_decrypt_ff1.o';

replace function encrypt_ff1_byteint (inputString byteint, inputKeyname
varchar(256)) returns varchar(8192) CHARACTER SET LATIN specific
encrypt_ff1_byteint language c no sql not deterministic parameter style sql called
on null input external name 'F:encrypt_ff1_byteint';

replace function encrypt_ff1_smallint (inputString smallint, inputKeyname
varchar(256)) returns varchar(8192) CHARACTER SET LATIN specific
encrypt_ff1_smallint language c no sql not deterministic parameter style sql called
on null input external name 'F:encrypt_ff1_smallint';
```

```
replace function encrypt_ff1_int (inputString integer, inputKeyname varchar(256))
returns varchar(8192) CHARACTER SET LATIN specific encrypt_ff1_int language c no
sql not deterministic parameter style sql called on null input external name
'F:encrypt_ff1_int';

replace function decrypt_ff1_byteint (inputString varchar(8192) CHARACTER SET
LATIN, inputKeyname varchar(256)) returns byteint specific decrypt_ff1_byteint
language c no sql not deterministic parameter style sql called on null input
external name 'F:decrypt_ff1_byteint';

replace function decrypt_ff1_smallint (inputString varchar(8192) CHARACTER SET
LATIN, inputKeyname varchar(256)) returns smallint specific decrypt_ff1_smallint
language c no sql not deterministic parameter style sql called on null input
external name 'F:decrypt_ff1_smallint';

replace function decrypt_ff1_int (inputString varchar(8192) CHARACTER SET LATIN,
inputKeyname varchar(256)) returns integer specific decrypt_ff1_int language c no
sql not deterministic parameter style sql called on null input external name
'F:decrypt_ff1_int';


* Comment out the following ten lines in order to run the ten UDFs in a separate
process.
* By default, UDFs run in a separate process ("protected mode"), which incurs a
performance penalty of up to 20x or 25x or 30x
alter function encrypt_string execute not protected;
alter function decrypt_data execute not protected;
alter function encrypt_char execute not protected;
alter function decrypt_char execute not protected;
alter function encrypt_cbc execute not protected;
alter function decrypt_cbc execute not protected;
alter function encrypt_fpe execute not protected;
alter function decrypt_fpe execute not protected;
alter function encrypt_ff1 execute not protected;
alter function decrypt_ff1 execute not protected;
alter function encrypt_byteint execute not protected;
alter function decrypt_byteint execute not protected;
alter function encrypt_smallint execute not protected;
alter function decrypt_smallint execute not protected;
alter function encrypt_int execute not protected;
alter function decrypt_int execute not protected;
alter function encrypt_time execute not protected;
alter function decrypt_time execute not protected;
alter function encrypt_date execute not protected;
alter function decrypt_date execute not protected;
alter function encrypt_fpe_byteint execute not protected;
alter function decrypt_fpe_byteint execute not protected;
alter function encrypt_fpe_smallint execute not protected;
alter function decrypt_fpe_smallint execute not protected;
```

```
alter function encrypt_fpe_int execute not protected;
alter function decrypt_fpe_int execute not protected;
alter function encrypt_ff1_byteint execute not protected;
alter function decrypt_ff1_byteint execute not protected;
alter function encrypt_ff1_smallint execute not protected;
alter function decrypt_ff1_smallint execute not protected;
alter function encrypt_ff1_int execute not protected;
alter function decrypt_ff1_int execute not protected;
.logoff
.quit
```

c. If you want to run in the Teradata Protected Mode, comment out the `alter function` lines. Leave them as is to run in the Teradata Unprotected Mode.

d. By default the BTEQ script installs the UDFs in the newly created database called `vormetric`. You may change the installation script to install them in the location of your choice.

e. Run the BTEQ script. Example:

```
# [/opt/vormetric/DataSecurityExpert/agent/pkcs11/teradata/udfs]#
./install_udfs.bteq
```

3. After this script has successfully run, the UDFs are installed in the newly created database with the default name vormetric.

Deploy Vormetric Protection on a Teradata Node

After installing Vormetric Protection for Teradata Database on the node, configure and deploy the application.

NOTE: This procedure must be done for every node.

1. Log in as root user on the node.
2. Optional: Set up VPTD white lists or black lists. Refer to "Managing User Access Control (Whitelist and Blacklist)" on page 9 and "Configuring Access Control" on page 21 to evaluate user access control options, and to determine how to leverage these features in your implementation.
3. Create a new profile configuration file based on the provided sample.

```
# cd /etc/vormetric
# cp profiles.conf.sample profiles.conf
```

Keep `profiles.conf.sample` as a reference, and use `profiles.conf` as your working configuration file. Modify this file to serve your purposes. For more information, see "encrypt_char()" on page 65.

4. Create a Vormetric Local Cryptoserver Daemon configuration file.

```
# cd /etc/vormetric
# cp vormetric_local_crypto_server.conf.sample
vormetric_local_crypto_server.conf
```

Keep `vormetric_local_crypto_server.conf.sample` as a reference, and use `vormetric_local_crypto_server.conf` as your working configuration file.

Modify the following lines in the Vormetric Local Cryptoserver Daemon configuration file `/etc/vormetric/vormetric_local_crypto_server.conf`:

```
Copyright (c) 2018 by <Your_Company>
iv <Initialization_Vector>
loglevel <desired_loglevel>
timeout <desired_whitelist-blacklist_polling_interval>
cryptoconf_timeout <in min>
udfaes <on_or_off>
```

where:

- ***Your_Company***: Name of your company.
- **iv:** Initialization vector, a 16-byte 32 hex character number. This must be the same for all nodes. Example: `0149AB83FC68D3C1395ABC4692DF931C`
- **loglevel**: Desired log level for logging events. Examples: `debug`, `info`, or `error` .
  - debug - Adds large, detailed debugging logs during crypto operations, with a very large performance overhead. A debug log level is *not* recommended in a production environment.
  - info - Adds less-informative logs when crypto operations are performed successfully, with reduced performance overhead. An info log level is *not* recommended in a production environment.
  - error - Adds logs only when an error occurs during product execution. The error log level is recommended in a production environment, because it incurs no performance overhead.
- **timeout**: The period, in minutes, between when the white list and black list are polled for changes and possible refresh. The default of 5 minutes should be adequate, but if you change your black/white list, you must wait up to 5 minutes before the changes take effect.
- **cryptoconf_timeout**: Length of time in minutes after which any new values are applied that have been added or changed since the last read of the configuration files. For example, the VPTD UDF cache for IV is cleared after this interval. Retrieving values every 5 or fewer minutes will incur some performance penalty during crypto operations. Only use a low value like 5 in a

test environment to assess the effect of changing values in configuration files (such as **iv** or **loglevel** values in the `vormetric_local_crypto_server.conf` file, or **iv** or **tweak** values in the `profies.conf` file).

NOTE: We recommended setting **cryptoconf_timeout** to `1440` minutes (1 day) to avoid any performance penalty in a production environment.

- **udfaes**: Specifies Normal Mode (`udfaes off`) or Fast Mode (`udfaes on`).

5. Invoke the Vormetric Local Cryptoserver with the `-e` command line option. Example:

   ```
   # /opt/vormetric/DataSecurityExpert/agent/pkcs11/teradata/bin/vormetric
   _local_crypto_server -e
   ```

   You are prompted for the PIN that you created in the last step of "Install Vormetric Protection for Teradata on the Teradata Node" on page 18. This PIN is written in encrypted form to the last line of `vormetric_local_crypto_server.conf`.

   NOTE: If you are using a DSM key manager (and have entered a DSM server hostname during installation), follow these remaining steps:

   If you are using a KeySecure key manager (and have entered a KeySecure server hostname during installation), continue by following the remaining steps 6 through 8 in the section "Configure VPTD with a KeySecure key manager" on page 37.

6. If you are using the Monit process supervision tool, then configure the `/etc/monitrc` file.

   Add these five lines:

   ```
   check process vormetric_local_crypto_server with pidfile
   /var/run/vormetric_local_crypto_server.pid
      start program = "/etc/init.d/vormetric_local_crypto_server start"
      stop  program = "/etc/init.d/vormetric_local_crypto_server stop"
      if failed unixsocket /tmp/vormetric then restart
      if 5 restarts within 5 cycles then timeout
   ```

7. Start the Vormetric Local Cryptoserver Daemon:

   ```
   # /etc/init.d/vormetric_local_crypto_server start
   ```

   Teradata database users now have access to the UDFs (see "VPTD UDFs" on page 41) and the VAE C API library.

8. Test the UDFs. In the following example, replace *SOMEPASSWORD* with your actual Teradata user password. The following example tests one of the UDFs. It is recommend to issue additional SQL requests to test all UDFs.

```
[/etc/vormetric]# bteq .logon vormetric,vormetric
Teradata BTEQ 14.10.00.10 for LINUX. PID: 11978
Copyright 1984-2014, Teradata Corporation. ALL RIGHTS RESERVED.
Enter your logon or BTEQ command:
.logon vormetric,SOMEPASSWORD
*** Logon successfully completed.
*** Teradata Database Release is 14.10.03.02
*** Teradata Database Version is 14.10.03.02
*** Transaction Semantics are BTET.
*** Session Character Set Name is 'ASCII'.
*** Total elapsed time was 2 seconds.
BTEQ -- Enter your SQL request or BTEQ command:
select vormetric.encrypt_string('hello world','KEY1');
select vormetric.encrypt_string('hello world','KEY1');
*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.
encrypt_string('hello world','KEY1')
--------------------------------------------------------------------------
490C35C33A5E07587ED4A6AFB15A16C9
```

Configure VPTD with a KeySecure key manager

If your node is using a SafeNet KeySecure key manager (and you entered a KeySecure server hostname during installation), perform the following additional steps after the initial installation.

1. Run the vormetric_local_crypto_server daemon located at `/opt/vormetric/DataSecurityExpert/agent/pkcs11/teradata/bin` location with `-e` option:

```
# cd /opt/vormetric/DataSecurityExpert/agent/pkcs11/teradata/bin
# vormetric_local_crypto_server -e
```

2. Enter the following configuration details:
   - IP Address of the KeySecure Server
   - NAE Port -- if not specified, defaults to 9000
   - KeySecure user name
   - KeySecure password
3. Enter NAE protocol, either `tcp` or `ssl`. **NOTE:** Using TCP is not secure.

- If you enter `tcp`, enter `y` at the following prompt:

```
Running registration will delete any existing CA and client
certificates. Do you wish to continue? <y/n>: y
#
```

VPTD configuration to KeySecure with TCP is complete.

- If you enter `ssl`, enter `y` at the following prompt:

```
Running registration will delete any existing CA and client
certificates. Do you wish to continue? <y/n>: y
```

You are prompted for additional information that is incorporqated into your certificate request:

- Country code (2 letter code, for example, US)
- State or Province name (for example, California)
- Locality or city name (for example, San Jose)
- Organization name (for example, company)
- Organizational Unit name (for example, section)
- Common Name (for example, your name or your server's hostname).

  Enter Common Name as a KeySecure login user-name when configuring the below NAE interface modes on KeySecure:

  - Verify client cert, user name taken from client cert, auth request is optional.

  - Verify client cert, password is needed, user name in cert must match user name in authentication request.

- Email Address (optional)

VPTD configuration to KeySecure with SSL is complete.

Automated Install over a Teradata Cluster

This feature detects a cluster and ensures that the installation is performed on all of the nodes in the cluster. The user does not have to do anything to make the installer look for the Cluster. This feature is built in. If the Cluster exists, the installer will find it. However, any change made in the configuration file is not propagated automatically on all nodes. It must be done manually on all of the nodes by the Admin.

When trying to install VPTD on a node, the installer detects the presence of the cluster and tries to install the same VPTD on the other nodes. This requires that the installation is performed in the “silent install” mode. This means that all requisite input is given in a file and VPTD is invoked with the -s option. If you use the Fingerprint method of registration, then the user must ensure that all of the nodes are added to the DSM prior to VPTD install.

Upgrade VPTD

The intent of this feature is to detect a cluster and ensure that the upgrade is performed on all of the nodes in the cluster.

NOTE: You must ensure that the crypto server daemon is not running on any single node before the uninstall begins.

When trying to upgrade the VPTD on a node, the installer detects the presence of the cluster and tries to upgrade the same VPTD on the other nodes. If there is a single node, then it upgrades the VPTD from that node as well.

You can upgrade VPTD from either v5.2.5.xx or from 6.1.3.xx to 6.4.0.xx. The upgrade procedure does not update the existing user modified configuration files. It adds the new configuration files, sample configuration files with new features, and binary files.

After the upgrade, you must follow the manual steps to remove the existing UDFs from the database and perform the new UDF installation.

Also, after the upgrade has successfully completed, you must restart the cryptographic server and flush the UDF cache using the **tpareset** command to clear old FF1 license information.

Warning! Failure to complete the cryptographic server restart and the UDF cache flush will cause the application to fail.

3

VPTD UDFs

Vormetric Protection for Teradata Database (VPTD) supports several user-defined functions (UDFs) to perform encryption and decryption operations on the Teradata database. This chapter contains the following sections:

Invoking the UDFs

To call one of the UDFs described in this chapter, you must invoke the Teradata BTEQ query tool using the following command:

ASCII login:

```
# bteq [.logon <UserID>,<Password>]
```

Unicode login:

```
# bteq -e UTF8 -c UTF16 [.logon <UserID>,<Password>]
```

You can then call the UDFs as follows:

```
BTEQ -- Enter your SQL request or BTEQ command:
select vormetric. <UDF name>(<UDF parameters>);
```

See the examples in the rest of this chapter.

You can also perform other BTEQ functions at the prompt. For more information about BTEQ, see Teradata documentation.

Mode Recommendations

VPTD UDFs are available in different general data mode categories:

- encrypt_cbc, encrypt_fpe, encrypt_ff1
- encrypt_char
- encrypt_string

The following are UDFs that support a Latin character set:

- encrypt_char/decrypt_char
- encrypt_string/decrypt_data

Note that encrypt_char takes an extra argument (column size) and it encrypts until the given column size, including any spaces present at the end of data in a column.

Usually, Teradata truncates spaces present at the end of input data, but VPTD is designed to encrypt spaces present after the input data.

It is not recommend to use encrypt_char/decrypt_char UDFs unless explicitly required.

The remaining are Unicode-supported UDFs (that is, they support both Unicode and Latin character sets).

- encrypt_cbc/decrypt_cbc
- encrypt_fpe/decrypt_fpe
- encrypt_ff1/decrypt_ff1

The fpe and ff1 UDFs are well-suited to preserve data format after encryption/decryption.

In general, it is recommend to use Unicode-supported UDFs instead of Latin ones.

encrypt_cbc()

Description

Given a cleartext string in Unicode format and a profile name (see "encrypt_char()" on page 65), returns its encrypted equivalent.

```
function encrypt_cbc (inputString varchar(8192) CHARACTER SET
UNICODE, inputKeyname varchar(256)) returns varbyte(16400)
```

The input string must adhere to BTEQ requirements, such as escaping special characters. For example, if the input string contains an apostrophe, use a double apostrophe: `'It''s a beautiful day'`.

Example

The following example encrypts a credit card number. This assumes that a profile has been set up in `profiles.conf` with the name `ccnum` and the encryption method `aes_cbc_pad`.

```
# select vormetric.encrypt_cbc('1234-9876-5678-6543', 'ccnum');
```

```
# select vormetric.encrypt_cbc('1234-9876-5678-6543', 'ccnum';
```

System Response

```
*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_cbc('1234-9876-5678-6543', 'ccnum'
--------------------------------------------------------------------
BE08C791A97E8FAC725DA39AFC071BD30A32C70C514E00B8D6D420501EF8B9D60F06537
CE93
```

decrypt_cbc()

Description

Given an encrypted string created using `encrypt_cbc()` and a profile name (see "encrypt_char()" on page 65), returns its decrypted cleartext.

```
function decrypt_cbc (inputString varbyte(16400),
  inputKeyname varchar(256)) returns varchar(8192) CHARACTER
  SET UNICODE
```

Example

The following example decrypts a credit card number. It is assumed that a profile has been set up in `profiles.conf` with the name `ccnum` and the encryption method `aes_cbc_pad`.

```
# BTEQ -- Enter your SQL request or BTEQ command:
select
vormetric.decrypt_cbc('BE08C791A97E8FAC725DA39AFC071BD30A32C70C514E00B8D6D420
501EF8B9D60F06537CE93'xb,'ccnum');

#select
vormetric.decrypt_cbc('BE08C791A97E8FAC725DA39AFC071BD30A32C70C514E00B8D6D420
501EF8B9D60F06537CE93'xb,'ccnum');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 1 second.

vormetric.decrypt_cbc('BE08C791A97E8FAC725DA39AFC071BD30A32C70C514E00B8D6
D420501EF8B9D60F06537CE93'xb,'ccnum')
--------------------------------------------------------------------
1234-9876-5678-6543
```

encrypt_int()

Description

Given a cleartext string in INTEGER dataytpe format and a keyname (see "encrypt_char()" on page 65), returns its encrypted equivalent.

```
function encrypt_int (inputString integer, inputKeyname
varchar(256)) returns varbyte(1600)
```

Example

The following example encrypts a negative integer number.

```
# BTEQ -- Enter your SQL request or BTEQ command:?
# select encrypt_int(-90000,'vptd-key');


# select encrypt_int(-90000,'vptd-key');
```

System Response

```
*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_int(-90000,'vptd-key');
----------------------------------------
6177A2C2E8D2491F48C01843136E88CB
```

decrypt_int()

Description

Given an encrypted string created using `encrypt_int()` and a keyname (see "encrypt_char()" on page 65), returns its decrypted cleartext.

```
function decrypt_int (inputString varbyte(16000),
  inputKeyname varchar(256)) returns integer
```

Example

The following example decrypts a negative integer number.

```
# BTEQ -- Enter your SQL request or BTEQ command:
#select decrypt_int('6177A2C2E8D2491F48C01843136E88CB'xb,'vptd-key');


#select decrypt_int('6177A2C2E8D2491F48C01843136E88CB'xb,'vptd-key');
```

```
vormetric.decrypt_int('6177A2C2E8D2491F48C01843136E88CB'xb,'vptd-key');
------------------------------------------------------------------------
-90000
```

encrypt_byteint()

Description

Given a cleartext string in BYTEINT datatype format and a keyname (see "encrypt_char()" on page 65), returns its encrypted equivalent.

```
function encrypt_byteint (inputdata byteint, inputKeyname
varchar(256)) returns varbyte(16000)
```

Example

The following example encrypts a byteint number.

```
# select encrypt_byteint(120,'vptd-key');


#select encrypt_byteint(120,'vptd-key');
```

System Response

```
*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_byteint(120,'vptd-key');
------------------------------------------
B66B4C6CB7F80416DD8FBA4A1211B5C4
```

decrypt_byteint()

Description

Given an encrypted string created using `encrypt_byteint()`and a keyname (see "encrypt_char()" on page 65), returns its decrypted cleartext.

```
function decrypt_byteint (inputString varbyte(16000),
  inputKeyname varchar(256)) returns byteint
```

Example

The following example decrypts a byteint number.

```
# BTEQ -- Enter your SQL request or BTEQ command:
#select decrypt_byteint('B66B4C6CB7F80416DD8FBA4A1211B5C4'xb,'vptd-key');


#select decrypt_byteint('B66B4C6CB7F80416DD8FBA4A1211B5C4'xb,'vptd-key');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 1 second.

vormetric.decrypt_byteint('B66B4C6CB7F80416DD8FBA4A1211B5C4'xb,'vptd-
key');
--------------------------------------------------------------------------
--
120
```

encrypt_smallint()

Description

Given a cleartext string in SMALLINT datatype format and a keyname (see "encrypt_char()" on page 65), returns its encrypted equivalent.

```
function encrypt_smallint (inputdata smallint, inputKeyname
varchar(256)) returns varbyte(16000)
```

Example

The following example encrypts a smallint number.

```
#select encrypt_smallint(32000,'vptd-key');


#select encrypt_smallint(32000,'vptd-key');
```

System Response

```
*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_smallint(32000,'vptd-key');
--------------------------------------------
E939997FBF75D6DCE47BBD1E1C1B69C7
```

decrypt_smallint()

Description

Given an encrypted string created using `encrypt_smallint()`and a keyname (see "encrypt_char()" on page 65), returns its decrypted cleartext.

```
function decrypt_smallint (inputString varbyte(16000),
  inputKeyname varchar(256)) returns smallint
```

Example

The following example decrypts a smallint number.

```
# BTEQ -- Enter your SQL request or BTEQ command:
#select decrypt_smallint('E939997FBF75D6DCE47BBD1E1C1B69C7'xb,'vptd-key');


#select decrypt_smallint('E939997FBF75D6DCE47BBD1E1C1B69C7'xb,'vptd-key');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 1 second.

vormetric.decrypt_smallint('E939997FBF75D6DCE47BBD1E1C1B69C7'xb,'vptd-
key');
--------------------------------------------------------------------------
-----
32000
```

encrypt_date()

Description

Given a cleartext string in DATE datatype format and a keyname (see "encrypt_char()" on page 65), returns its encrypted equivalent.

```
function encrypt_date (inputdata date, inputKeyname varchar(256))
returns varbyte(16000)
```

The input string must adhere to BTEQ requirements, such as escaping special characters. For example, if the input string contains an apostrophe, use a double apostrophe: `'It''s a beautiful day'`.

Example

The following example encrypts a date number.

```
#select encrypt_date(DATE '2019-03-20','vptd-key');
```

```
#select encrypt_date(DATE '2019-03-20','vptd-key');
```

System Response

```
*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_date(DATE '2019-03-20','vptd-key');
--------------------------------------------------
F8BD65A76B9B5D5ED135CDE7F8E2DB13
```

decrypt_date()

Description

Given an encrypted string created using `encrypt_date()` and a kayname (see "encrypt_char()" on page 65), returns its decrypted cleartext.

```
function decrypt_date (inputString varbyte(16000),
  inputKeyname varchar(256)) returns date
```

Example

The following example decrypts a date number.

```
# BTEQ -- Enter your SQL request or BTEQ command:
#select decrypt_date('F8BD65A76B9B5D5ED135CDE7F8E2DB13'xb,'vptd-key');


#select decrypt_date('F8BD65A76B9B5D5ED135CDE7F8E2DB13'xb,'vptd-key');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 1 second.

vormetric.decrypt_date('F8BD65A76B9B5D5ED135CDE7F8E2DB13'xb,'vptd-key')
--------------------------------------------------------------------------------
-----
19/03/20
```

encrypt_time()

Description

Given a cleartext string in TIME datatype format and a keyname (see "encrypt_char()" on page 65), returns its encrypted equivalent.

```
function encrypt_time (inputdata time, inputKeyname varchar(256))
returns varbyte(16000)
```

The input string must adhere to BTEQ requirements, such as escaping special characters. For example, if the input string contains an apostrophe, use a double apostrophe: `'It''s a beautiful day'`.

Example

The following example encrypts a time number. T

```
#select encrypt_time(TIME '11:37:58','vptd-key');


#select encrypt_time(TIME '11:37:58','vptd-key');
```

System Response

```
*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_time(TIME '11:37:58','vptd-key')
--------------------------------------------------
455F982498C2D473B920FF26124832C2
```

decrypt_time()

Description

Given an encrypted string created using `encrypt_time()` and a keyname (see "encrypt_char()" on page 65), returns its decrypted cleartext.

```
function decrypt_time (inputString varbyte(16400),
 inputKeyname varchar(256)) returns varchar(8192) CHARACTER
 SET UNICODE
```

Example

The following example decrypts a time number.

```
# BTEQ -- Enter your SQL request or BTEQ command:
#select decrypt_time('455F982498C2D473B920FF26124832C2'xb,'vptd-key');
```

```
#select decrypt_time('455F982498C2D473B920FF26124832C2'xb,'vptd-key');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 1 second.

vormetric.decrypt_time('455F982498C2D473B920FF26124832C2'xb,'vptd-key');
--------------------------------------------------------------------------------
-----
11:37:58.0000
```

encrypt_timestamp()

Description

Given a cleartext string in TIMESTAMP datatype format and a keyname (see "encrypt_char()" on page 65), returns its encrypted equivalent.

```
function encrypt_timestamp (inputdata timestamp, inputKeyname
varchar(256)) returns varbyte(16000)
```

The input string must adhere to BTEQ requirements, such as escaping special characters. For example, if the input string contains an apostrophe, use a double apostrophe: `'It''s a beautiful day'`.

Example

The following example encrypts a timestamp number.

```
#select encrypt_timestamp(TIMESTAMP '2020-03-20 19:22:05', 'vptd-key');


#select encrypt_timestamp(TIMESTAMP '2020-03-20 19:22:05', 'vptd-key');
```

System Response

```
*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_timestamp(TIMESTAMP '2020-03-20 19:22:05', 'vptd-
key')
--------------------------------------------------------------------------------
---
E4B5E6FD2C86E2B8F41A92F554ECAA02B5007986DB21BF1A1D95D24F80064107
```

decrypt_timestamp()

Description

Given an encrypted string created using `encrypt_timestamp()`and a keyname (see "encrypt_char()" on page 65), returns its decrypted cleartext.

```
function decrypt_timestamp (inputString varbyte(16000),
  inputKeyname varchar(256)) returns timestamp
```

Example

The following example decrypts a timestamp number.

```
# BTEQ -- Enter your SQL request or BTEQ command:
#select
decrypt_timestamp('E4B5E6FD2C86E2B8F41A92F554ECAA02B5007986DB21BF1A1D95D24F80
064107'xb,'vptd-key');


#select
decrypt_timestamp('E4B5E6FD2C86E2B8F41A92F554ECAA02B5007986DB21BF1A1D95D24F80
064107'xb,'vptd-key');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 1 second.

vormetric.decrypt_timestamp('E4B5E6FD2C86E2B8F41A92F554ECAA02B5007986DB21
BF1A1D95D24F80064107'xb,'vptd-key')
--------------------------------------------------------------------------
----------------------------------

2020-03-20 19:22:05.000000
```

encrypt_fpe()

Description

Given a cleartext string in Unicode or Latin characters and a profile name (see "encrypt_char()" on page 65), returns its encrypted equivalent.

```
function encrypt_fpe (inputString varchar(8192) CHARACTER
  SET UNICODE, inputKeyname varchar(256)) returns
  varchar(8192) CHARACTER SET UNICODE
```

To accept Latin characters as input, the `encrypt_fpe()`call must refer to a profile that uses a predefined Latin character set. For more information, see "Using profiles.conf with FPE" on page 68.

If there are characters in the plain text input that are not specified in the character set, they are left in their current positions, unchanged, in the tokenized output. If the plain text input is less than 2 characters (not counting characters that are not specified in the character set), the output of the UDF is the same as the plaintext input. FPE can tokenize only strings with 2 or more characters.

The input string must adhere to BTEQ requirements, such as escaping special characters. For example, if the input string contains an apostrophe, use a double apostrophe: `'It''s a beautiful day'`.

Example

The following example encrypts a customer's first and last name. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_name` and the encryption method `fpe`.

```
# select vormetric.encrypt_fpe('John Doe','tokenize_name');

# select vormetric.encrypt_fpe('John Doe','tokenize_name');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_fpe('John Doe','tokenize_name')
--------------------------------------------------------------------
OHBU cCj
```

decrypt_fpe()

Description

Given an encrypted string created using `encrypt_fpe()`and a profile name, returns its decrypted cleartext.

```
function decrypt_fpe (inputString varchar(8192) CHARACTER
  SET UNICODE, inputKeyname varchar(256)) returns
  varchar(8192) CHARACTER SET UNICODE
```

Example

The following example decrypts a customer's first and last name. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_name` and the encryption method `fpe`.

```
BTEQ -- Enter your SQL request or BTEQ command:
  # select vormetric.decrypt_fpe('OHBU cCj','tokenize_name');
```

```
# select vormetric.decrypt_fpe('OHBU cCj','tokenize_name');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 1 second.

vormetric.decrypt_fpe('OHBU cCj','tokenize_name')
------------------------------------------------------------------------
John Doe
```

encrypt_fpe_int()

Description

Given a cleartext string in the INTEGER datatype and a profile name, returns its encrypted equivalent.

```
function encrypt_fpe_int (inputString integer, inputKeyname
  varchar(256)) returns varchar(8192) CHARACTER SET LATIN
```

If there are characters in the plain text input that are not specified in the character set, they are left in their current positions, unchanged, in the tokenized output. If the plain text input is less than 2 characters (not counting characters that are not specified in the character set), the output of the UDF is the same as the plaintext input. FPE can tokenize only strings with 2 or more characters.

Example

The following example encrypts a negative number. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_CC` and the encryption method `fpe`.

```
#select encrypt_fpe_int(-90000,'tokenize_CC');


#select encrypt_fpe_int(-90000,'tokenize_CC');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_fpe_int(-90000,'tokenize_CC')
------------------------------------------------------------------
-65048
```

decrypt_fpe_int()

Description

Given an encrypted string created using `encrypt_fpe_int()`and a profile name, returns its decrypted cleartext.

```
function decrypt_fpe_int (inputString varchar(8192)
  CHARACTER SET LATIN, inputKeyname varchar(256)) returns
  integer
```

Example

The following example decrypts a negative number. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_CC` and the encryption method `fpe`.

```
BTEQ -- Enter your SQL request or BTEQ command:
  #select decrypt_fpe_int('-65048','tokenize_CC');


  #select decrypt_fpe_int('-65048','tokenize_CC');

  *** Query completed. One row found. One column returned.
  *** Total elapsed time was 1 second.

  vormetric.decrypt_fpe_int('-65048','tokenize_CC')
  ------------------------------------------------
  -90000
```

encrypt_fpe_byteint()

Description

Given a cleartext string in the BYTEINT datatype and a profile name, returns its encrypted equivalent.

```
function encrypt_fpe_byteint (inputString byteint,
  inputKeyname varchar(256)) returns varchar(8192) CHARACTER
  SET LATIN
```

If there are characters in the plain text input that are not specified in the character set, they are left in their current positions, unchanged, in the tokenized output. If the plain text input is less than 2 characters (not counting characters that are not specified in the character set), the output of the UDF is the same as the plaintext input. FPE can tokenize only strings with 2 or more characters.

Example

The following example encrypts a byteint number. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_CC` and the encryption method `fpe`.

```
#select encrypt_fpe_byteint(120,'tokenize_CC');


#select encrypt_fpe_byteint(120,'tokenize_CC');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_fpe_byteint(120,'tokenize_CC')
------------------------------------------------
207
```

decrypt_fpe_byteint()

Description

Given an encrypted string created using `encrypt_fpe_byteint()`and a profile name, returns its decrypted cleartext.

```
function decrypt_fpe_byteint (inputString varchar(8192)
  CHARACTER SET LATIN, inputKeyname varchar(256)) returns
  byteint
```

Example

The following example decrypts a byteint number. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_CC` and the encryption method `fpe`.

```
BTEQ -- Enter your SQL request or BTEQ command:
  #select decrypt_fpe_byteint('207','tokenize_CC');


  #select decrypt_fpe_byteint('207','tokenize_CC');

  *** Query completed. One row found. One column returned.
  *** Total elapsed time was 1 second.

  vormetric.decrypt_fpe_byteint('207','tokenize_CC')
  ---------------------------------------------------------------------
  120
```

encrypt_fpe_smallint()

Description

Given a cleartext string in the SMALLINT datatype and a profile name, returns its encrypted equivalent.

```
function encrypt_fpe_smallint (inputString smallint,
  inputKeyname varchar(256)) returns varchar(8192) CHARACTER
  SET LATIN
```

If there are characters in the plain text input that are not specified in the character set, they are left in their current positions, unchanged, in the tokenized output. If the plain text input is less than 2 characters (not counting characters that are not specified in the character set), the output of the UDF is the same as the plaintext input. FPE can tokenize only strings with 2 or more characters.

Example

The following example encrypts a smallint number. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_CC` and the encryption method `fpe`.

```
#select encrypt_fpe_smallint(21690,'tokenize_CC');


#select encrypt_fpe_smallint(21690,'tokenize_CC');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_fpe_smallint('21690','tokenize_CC')
------------------------------------------------------------------------
13490
```

decrypt_fpe_smallint()

Description

Given an encrypted string created using `encrypt_fpe_smallint()`and a profile name, returns its decrypted cleartext.

```
function decrypt_fpe_smallint (inputString varchar(8192)
  CHARACTER SET LATIN, inputKeyname varchar(256)) returns
  smallint
```

Example

The following example decrypts a smallint number. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_CC` and the encryption method `fpe`.

```
BTEQ -- Enter your SQL request or BTEQ command:
  #select decrypt_fpe_smallint('13490','tokenize_CC');


  #select decrypt_fpe_smallint('13490','tokenize_CC');

  *** Query completed. One row found. One column returned.
  *** Total elapsed time was 1 second.

  vormetric.decrypt_fpe('13490','tokenize_CC')
  ----------------------------------------------------------------------
  21690
```

encrypt_ff1()

Description

Given a cleartext string in Unicode or Latin characters and a profile name (see "encrypt_char()" on page 65), returns its encrypted equivalent.

```
function encrypt_ff1 (inputString varchar(8192) CHARACTER
  SET UNICODE, inputKeyname varchar(256)) returns
  varchar(8192) CHARACTER SET UNICODE
```

If there are characters in the plain text input that are not specified in the character set, they are left in their current positions unchanged in the tokenized output. If the plain text input is less than 2 characters (not counting characters that are not specified in the character set), the output of the UDF is the same as the plain text input. FF1 can tokenize only strings with 2 or more characters.

Example

The following example encrypts a customer's first and last name. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_name` and the encryption method `ff1`.

```
  # select vormetric.encrypt_ff1('John Doe','tokenize_name');

  # select vormetric.encrypt_ff1('John Doe','tokenize_name');

  *** Query completed. One row found. One column returned.
  *** Total elapsed time was 2 seconds.

  vormetric.encrypt_ff1('John Doe','tokenize_name')
```

```
------------------------------------------------------------------
OHBU cCj
```

decrypt_ff1()

Description

Given an encrypted string created using `encrypt_ff1()`and a profile name, returns its decrypted cleartext.

```
function decrypt_ff1 (inputString varchar(8192) CHARACTER
  SET UNICODE, inputKeyname varchar(256)) returns
  varchar(8192) CHARACTER SET UNICODE
```

Example

The following example decrypts a customer's first and last name. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_name` and the encryption method `ff1`.

```
BTEQ -- Enter your SQL request or BTEQ command:
  # select vormetric.decrypt_ff1('OHBU cCj','tokenize_name');

  # select vormetric.decrypt_ff1('OHBU cCj','tokenize_name');

  *** Query completed. One row found. One column returned.
  *** Total elapsed time was 1 second.

  vormetric.decrypt_ff1('OHBU cCj','tokenize_name')
  ------------------------------------------------------------------------
  John Doe
```

encrypt_ff1_int()

Description

Given a cleartext string in INT datatype and a profile name, returns its encrypted equivalent.

```
function encrypt_ff1_int (inputString integer, inputKeyname
  varchar(256)) returns varchar(8192) CHARACTER SET LATIN
```

If there are characters in the plain text input that are not specified in the character set, they are left in their current positions unchanged in the tokenized output. If the plain text input is less than 2 characters (not counting characters that are not specified in the character set), the

output of the UDF is the same as the plain text input. FF1 can tokenize only strings with 2 or more characters.

Example

The following example encrypts an integer number. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_CC_FF1` and the encryption method `ff1`.

```
#select encrypt_ff1_int(-90000,'tokenize_CC_FF1');


#select encrypt_ff1_int(-90000,'tokenize_CC_FF1');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_ff1_int('-90000','tokenize_CC_FF1')
------------------------------------------------------------------
99473
```

decrypt_ff1_int()

Description

Given an encrypted string created using `encrypt_ff1_int()`and a profile name, returns its decrypted cleartext.

```
function decrypt_ff1_int (inputString varchar(8192)
  CHARACTER SET LATIN, inputKeyname varchar(256)) returns
  integer
```

Example

The following example decrypts an integer number. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_CC_FF1` and the encryption method `ff1`.

```
BTEQ -- Enter your SQL request or BTEQ command:
  #select decrypt_ff1_int('-99473','tokenize_CC_FF1');


  #select decrypt_ff1_int('-99473','tokenize_CC_FF1');

  *** Query completed. One row found. One column returned.
  *** Total elapsed time was 1 second.

  vormetric.decrypt_ff1_int('-99473','tokenize_CC_FF1')
  ------------------------------------------------------------------
  -90000
```

encrypt_ff1_byteint()

Description

Given a cleartext string in BTYTEINT datatype and a profile name, returns its encrypted equivalent.

```
function encrypt_ff1_byteint (inputString byteint,
  inputKeyname varchar(256)) returns varchar(8192) CHARACTER
  SET LATIN
```

If there are characters in the plain text input that are not specified in the character set, they are left in their current positions unchanged in the tokenized output. If the plain text input is less than 2 characters (not counting characters that are not specified in the character set), the output of the UDF is the same as the plain text input. FF1 can tokenize only strings with 2 or more characters.

Example

The following example encrypts a byteint number. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_CC_FF1` and the encryption method `ff1`.

```
#select encrypt_ff1_byteint(120,'tokenize_CC_FF1');


#select encrypt_ff1_byteint(120,'tokenize_CC_FF1');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 2 seconds.

vormetric.encrypt_ff1_byteint('120','tokenize_CC_FF1')
------------------------------------------------------------------------
577
```

decrypt_ff1_byteint()

Description

Given an encrypted string created using `encrypt_ff1_byteint()`and a profile name, returns its decrypted cleartext.

```
function decrypt_ff1_byteint (inputString varchar(8192)
  CHARACTER SET LATIN, inputKeyname varchar(256)) returns
  byteint
```

Example

The following example decrypts a byteint number. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_CC_FF1` and the encryption method `ff1`.

```
BTEQ -- Enter your SQL request or BTEQ command:
  #select decrypt_ff1_byteint('577','tokenize_CC_FF1');


  #select decrypt_ff1_byteint('577','tokenize_CC_FF1');

  *** Query completed. One row found. One column returned.
  *** Total elapsed time was 1 second.

  vormetric.decrypt_ff1_byteint('577','tokenize_CC_FF1')
  ------------------------------------------------------------------------
  120
```

encrypt_ff1_smallint()

Description

Given a cleartext string in SMALLINT datatype and a profile name, returns its encrypted equivalent.

```
function encrypt_ff1_smallint (inputString smallint,
  inputKeyname varchar(256)) returns varchar(8192) CHARACTER
  SET LATIN
```

If there are characters in the plain text input that are not specified in the character set, they are left in their current positions unchanged in the tokenized output. If the plain text input is less than 2 characters (not counting characters that are not specified in the character set), the output of the UDF is the same as the plain text input. FF1 can tokenize only strings with 2 or more characters.

Example

The following example encrypts a snallint number. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_CC_FF1` and the encryption method `ff1`.

```
  #select encrypt_ff1_smallint(21690,'tokenize_CC_FF1');


  #select encrypt_ff1_smallint(21690,'tokenize_CC_FF1');

  *** Query completed. One row found. One column returned.
  *** Total elapsed time was 2 seconds.
```

```
vormetric.encrypt_ff1_smallint('21690','tokenize_CC_FF1')
-----------------------------------------------------------------------
83365
```

decrypt_ff1_smallint()

Description

Given an encrypted string created using `encrypt_ff1_smallint()`and a profile name, returns its decrypted cleartext.

```
function decrypt_ff1_smallint (inputString varchar(8192)
  CHARACTER SET LATIN, inputKeyname varchar(256)) returns
  smallint
```

Example

The following example decrypts a smallint number. It is assumed that a profile has been set up in `profiles.conf` with the name `tokenize_CC_FF1` and the encryption method `ff1`.

```
BTEQ -- Enter your SQL request or BTEQ command:
  #select decrypt_ff1_smallint('83365','tokenize_CC_FF1');


  #select decrypt_ff1_smallint('83365','tokenize_CC_FF1');

  *** Query completed. One row found. One column returned.
  *** Total elapsed time was 1 second.

  vormetric.decrypt_ff1_smallint('83365','tokenize_CC_FF1')
  -----------------------------------------------------------------------
  21690
```

encrypt_string()

NOTE: The `encrypt_cbc()`, `encrypt_fpe(,)` and `encrypt_ff1()` UDFs are recommended. Use of `encrypt_string()` is not recommended.

Description

Given a cleartext string in Latin characters and a key, returns its encrypted equivalent.

```
function encrypt_string (inputString varchar(16384),
  inputKeyname varchar(256)) returns varbyte(16400)
```

The input string must adhere to BTEQ requirements, such as escaping special characters. For example, if the input string contains an apostrophe, use a double apostrophe: `'It''s a beautiful day'`.

NOTE: The initialization vector (IV) for `encrypt_string()` is taken from the file `/etc/vormetric/vormetric_local_crypto_server.conf`.

Example

```
BTEQ -- Enter your SQL request or BTEQ command:
  # select vormetric.encrypt_string('clear text', 'KEY1');
  # select vormetric.encrypt_string('clear text', 'KEY1');

  *** Query completed. One row found. One column returned.
  *** Total elapsed time was 2 seconds.

  vormetric.encrypt_string('clear text','KEY1')
  ----------------------------------------------------------------------
  97E7BCEB7D8EA55A59BC83BD44609B0A
```

decrypt_data()

Description

Given an encrypted string created using `encrypt_string()` and the key used to encrypt data, returns its decrypted cleartext.

```
function decrypt_data (inputString varbyte(16400),
  inputKeyname varchar(256)) returns varchar(16384)
```

NOTE: Cannot be used to decrypt data that was created using `encrypt_cbc()`, `encrypt_fpe()`, or `encrypt_ff1()`. Use the corresponding `decrypt_cbc()`, `decrypt_fpe()`, or `decrypt_ff1()` functions instead.

Example

```
BTEQ -- Enter your SQL request or BTEQ command:
# select vormetric.decrypt_data('97E7BCEB7D8EA55A59BC83BD44609B0A'xb,'KEY1');

# select vormetric.decrypt_data('97E7BCEB7D8EA55A59BC83BD44609B0A'xb,'KEY1');

*** Query completed. One row found. One column returned.
*** Total elapsed time was 1 second.

vormetric.decrypt_data('97E7BCEB7D8EA55A59BC83BD44609B0A'XB,'KEY1')
--------------------------------------------------------------------------------
clear text
```

encrypt_char()

Description

Given a cleartext string in Latin characters, a key and column size returns its encrypted equivalent.

```
function encrypt_char (inputString varchar(16384), inputKeyname
varchar(256), inputCharcolumnsize INTEGER) returns varbyte(16000)
```

NOTE: The initialization vector (IV) for `encrypt_char( )` is taken from the file `/etc/vormetric/vormetric_local_crypto_server.conf`. A sample IV = 000102030405060708090A0B0C0D0E0F.

Example

```
CREATE SET TABLE VORMETRIC.testvar2 ,NO FALLBACK ,
     NO BEFORE JOURNAL,
     NO AFTER JOURNAL,
     CHECKSUM = DEFAULT,
     DEFAULT MERGEBLOCKRATIO
     (column1 INTEGER,
      column2 CHAR(13)
     );
```

```
BTEQ -- Enter your SQL request or BTEQ command:


select  column1,vormetric.encrypt_char(column2, 'KEY-NEW' ,13) as
enc_col2 from testvar2;
```

```
select  column1,vormetric.encrypt_char(column2, 'KEY-NEW' ,13) as
enc_col2 from testvar2;


  *** Query completed. 3 rows found. 2 columns returned.
  *** Total elapsed  time  was 1 second.
     column1 enc_col2
----------- ----------------------------------------------------
            3 FB8CAA2F658963C73233A59589633B22
            1 4C69855E490704EFF93DFEED957201F3
            2 5F252180E7F86AE27F7050E3CE514115
```

decrypt_char()

Description

Given an encrypted string created using `encrypt_char()`, the key used to encrypt data, and the column size, returns its decrypted cleartext.

```
function decrypt_char (inputString varbyte(16000), inputKeyname varchar(256),
inputCharcolumnsize INTEGER) returns varchar(16384)
```

NOTE: Cannot be used to decrypt data that was created using `encrypt_string()`, `encrypt_cbc()`, `encrypt_fpe()`, or `encrypt_ff1()`. Use the corresponding `decrypt_data()`, `decrypt_cbc()` `decrypt_fpe()`, or `decrypt_ff1()` with those UDFs instead.

Example

See the encrypt_char example code to create a table.

```
BTEQ -- Enter your SQL request or BTEQ command:


select  column1, '['  || vormetric.decrypt_char(
vormetric.encrypt_char(column2, 'KEY-NEW' ,13) , 'KEY-NEW' ,13)
||  ']'   as enc_dec_col2 from testvar2;
```

```
select  column1, '['  || vormetric.decrypt_char(
vormetric.encrypt_char(column2, 'KEY-NEW' ,13) , 'KEY-NEW' ,13)
||  ']'   as enc_dec_col2 from testvar2;

  *** Query completed. 3 rows found. 2 columns returned.

  *** Total elapsed  time  was 1 second.

     column1 enc_dec_col2
 ----------- ------------------------------------------------------
           3 [1236974      ]

           1 [1234567890123]

           2 [rt           ]
```

NOTE: Column size is maintained and filled with trailing spaces in the output, which would not be possible using `encrypt_string/decrypt_data`.

profiles.conf

To streamline the invocation of the `encrypt_cbc()`, `decrypt_cbc()`, `encrypt_fpe()`, `decrypt_fpe()`, `encrypt_ff1()`, and `decrypt_ff1()` UDFs, the file `profiles.conf` is used. This file contains named profiles that include several values required as input to these UDFs. Provide the profile name as a parameter when invoking the UDF. The UDF looks up the profile name and uses the parameters that are grouped under that profile name.

An example file named `profiles.conf.sample` is provided with the installation software. Make your own copy of this file and change the name to `profiles.conf`.

Be sure to set the file permissions for `profiles.conf` appropriately. The file should be owned by the root user, who can create and edit the file. Other users should be able to view the file, so they can see the profile names in order to reference them in their own UDF calls. It is not recommended to allow users other than root to edit `profiles.conf`. Set the file permissions to 644.

NOTE: Profiles are not supported for the `encrypt_string()` and `decrypt_data()` UDFs.

Using profiles.conf with Unicode Block Cipher UDFs

Profiles that are to be used with the UDFs `encrypt_cbc()` and `decrypt_cbc()` must contain the following key-value pairs:

```
method = aes_cbc_pad
iv = Initialization Vector Number
keyname = Name of the Vormetric encryption key to use
```

The following is a sample `profiles.conf` file that contains two profiles named `ccnum` and `address`:

```
[ccnum]
method = aes_cbc_pad
iv = 000102030405060708090A0B0C0D0E0F
keyname = KEY_1
[address]
method = aes_cbc_pad
iv = 0F0E0D0C0B0A09080706050403020100
keyname = KEY_1
```

With these profile definitions, the following UDF calls can be made:

```
encrypt_cbc('1234-9876-5678-6543', 'ccnum')
encrypt_cbc('2860 Junction Avenue, San Jose, CA 95134', 'address')
```

Using profiles.conf with FPE

Profiles that are to be used with the UDFs `encrypt_fpe()` and `decrypt_fpe()` must contain the following key-value pairs:

```
method = fpe
tweak = 8-byte tweak | auto
charset = One of the values specified in [character_sets]
keyname = Name of the Vormetric encryption key to use
```

If `tweak` is set to `auto`, the tweak is generated automatically for you.

In the `[character_sets]` section of `profiles.conf`, specify one or more named character sets which can be used to tokenize the cleartext input. Define each character set as one or

more comma-separated Unicode ranges in UTF16 Big Endian format. Then, in `charset`, provide one of the names from this section.

Also note that FPE-related UDFs can optionally set the parameter `allowSingleCharInputs` to `yes` to permit passing through any NULL or single-character input without encrypting it, while the remaining data are encrypted as expected.

Example

The following is a sample `profiles.conf` file for use with `encrypt_fpe()` and `decrypt_fpe()`. It defines several character sets, including a Latin character set, then defines two profiles named `tokenize_name` and `tokenize_address`,

```
[character_sets]
latin = 0020-007E
alphanumeric = 0030-0039,0041-005A,0061-007A
hindi = 0905-0939,0958-0961
custom = 0905-0939,0030-0039
[tokenize_name]
method = fpe
tweak = D8E7920AFA330A73
charset = latin
keyname = KEY_FPE_1
[tokenize_address]
method = fpe
tweak = auto
charset = alphanumeric
keyname = KEY_FPE_1
```

With these profile definitions, the following UDF calls can be made:

```
encrypt_fpe('John Doe', 'tokenize_name')
encrypt_fpe('2860 Junction Avenue, San Jose, CA 95134', 'tokenize_address')
```

Using profiles.conf with FF1

Profiles that are to be used with the UDFs `encrypt_ff1()` and `decrypt_ff1()` must contain the following key-value pairs:

```
method = ff1
tweak = 8-byte tweak | auto
charset = One of the values specified in [character_sets]
keyname = Name of the Vormetric encryption key to use
```

If tweak is set to auto, the tweak is generated automatically for you.

In the [character_sets] section of profiles.conf, specify one or more named character sets which can be used to tokenize the cleartext input. Define each character set as one or more comma-separated Unicode ranges in UTF16 Big Endian format. Then, in charset, provide one of the names from this section.

Also note that FF1-related UDFs can optionally set the parameter allowSingleCharInputs to yes to permit passing through any NULL or single-character input without encrypting it, while the remaining data are encrypted as expected.

Example

The following is a sample profiles.conf file for use with encrypt_ff1() and decrypt_ff1(). It defines several character sets, including a Latin character set, then defines two profiles named tokenize_name and tokenize_address,

```
[character_sets]
latin = 0020-007E
alphanumeric = 0030-0039,0041-005A,0061-007A
hindi = 0905-0939,0958-0961
custom = 0905-0939,0030-0039
[tokenize_name]
method = ff1
tweak = D8E7920AFA330A73
charset = latin
keyname = KEY_FF1_1
[tokenize_address]
method = ff1
tweak = auto
charset = alphanumeric
keyname = KEY_FF1_1
```

With these profile definitions, the following UDF calls can be made:

```
encrypt_ff1('John Doe', 'tokenize_name')
encrypt_ff1('2125 Zanker Avenue, San Jose, CA 95131', 'tokenize_address')
```

Allowing Single Character or NULL Inputs

Untranslatable character errors sometimes occur when passing in data to FPE- or FF1-related UDFs that include NULL values or values with insufficient characters to encrypt. You can avoid this error by using the configuration parameter allowSingleCharInputs.

When you set the parameter `allowSingleCharInputs` to `yes` in the `profiles.conf` file, VPTD passes through any NULL or single-character input without encrypting it, while the remaining data are encrypted as expected.

Data Masking for Encryption and Decryption

Masking allows the VPTD administrator to define one mask per user. The masking rules trigger during decryption with the UDF `decrypt_fpe()`. The following sections describe the following related features:

- Define masks
- Support partial encryption capability
- Add prefix and suffix to an encrypted value
- Specify a field's encryption as irreversible
- Provide encrypted data that can pass a Luhn check

Defining Masks

Masks are defined using the configuration file: `/etc/vormetric/masks.conf`. The format of `masks.conf` is similar to that of `profiles.conf`. It is an `.ini` file with sections and key-value pairs. The section name represents the Teradata username. Each section contains the following key value pairs representing the mask definition:

- **Showfirst**: A numeric value representing the number of characters at the beginning which display in the decrypted output. By default, this value is zero.

  For example, a `showfirst` value of 4 means that the output would look like 3454-XXXX-XXXX-XXXX.

- **Showlast**: A numeric value representing the number of characters at the end which display in the decrypted output. By default, this value is zero.

  For example, a `showlast` value of 4 means that the output would look like XXXX-XXXX-XXXX-8462

- **Maskchar**: The masking character used in the decrypted output. This is mandatory for a valid mask definition.

Warning! If `masks.conf` is not found, or a username is not found in `masks.conf`, then the decrypted output will be shown in its entirety.

Example

```
# A sample masks.conf
[DBC]
showfirst = 4
showlast = 4
maskchar = 'X'
[VORMETRIC]
showfirst = 0
showlast = 0
maskchar = '*'
```

Supporting Partial Encryption

You can have partial encryption on the selected field value. This is governed by the profile given as a parameter to the UDF. The following key-value pairs allow partial encryption:

- **Keepleft**: A numeric value denoting the number of characters which will remain as they are at the beginning of the input string, while the rest of the string is encrypted.

  For example, a `keepleft` value of 4 for input 1234-5678-3456-7890 results in output similar to: 1234-6573-7412-8831.

- **Keepright**: A numeric value denoting the number characters which will remain as they are at the end of the input string, while the rest of the string is encrypted.

  For example, a `keepright` value of 4 for input 1234-5678-3456-7890 might result in output similar to: 6438-6573-7412-7890.

  Example

  ```
  [encrypt_ccnum]
  method = fpe
  tweak = auto
  charset = digits
  keyname = KEY_FPE_CC
  keepleft = 4
  ```

Adding Prefix and Suffix to a Decrypted Field Value

You can add a prefix and/or suffix to the decrypted output through a new key-value pair: **prefix** and **suffix**, using the `profiles.conf` file. If the prefix is defined as a string "Credit Card number: " then the output results in output similar to: Credit Card number: 1234-6352-1738-2343.

The size of the prefix/suffix is limited to 10 characters.

Example:

```
[encrypt_ccnum]
method = fpe
tweak = auto
charset = digits
keyname = KEY_FPE_CC
keepleft = 4
prefix = "Credit Card number: "
```

Generating an Irreversible Encryption

If you need to encrypt data that can never be decrypted:

1. Use the key **irreversible**.
2. Set it to **yes** to encrypt forever.

   Example

   ```
   [irreversible_encryption]
   method = fpe
   tweak = auto
   charset = digits
   keyname = KEY_FPE_CC
   irreversible = yes
   ```

Luhn-check-compatible Encryption

A Luhn check is a checksum algorithm used to verify if a credit card number is valid. Users use FPE so that legacy applications do not break while processing the encrypted data. Running a standard FPE for a credit card number might result in a string which looks like a credit card number, but might fail the Luhn check. VPTD supports format-preserving encryption that will pass the Luhn check. For this, VPTD had added a method called `fpeluhn` in `profiles.conf`. The charset for this profile is ASCII digits, and the keepright directive will not work with this encryption method.

FPE-Luhn in Teradata requires three or more characters for encryption.

```
[encrypt_ccnum]
method = fpeluhn \\ New method to enforce Luhn check validation for the output
tweak = auto
charset = digits
keyname = KEY_FPE_CC
```

4 Teradata MultiLoad and FastExport

Vormetric Protection for Teradata Database (VPTD) supports Teradata MultiLoad and FastExport utilities.

About MultiLoad

Teradata MultiLoad is a command-driven utility for fast, high-volume maintenance on multiple tables and views in a Teradata database.

A single Teradata MultiLoad job can perform several different *import* and *delete* tasks on database tables and views:

- Each Teradata MultiLoad *import* task can do multiple data insert, update, and delete functions on up to five different tables or views.
- Each Teradata MultiLoad *delete* task can remove large numbers of rows from a single table.

The Teradata MultiLoad utility processes a series of commands and Teradata SQL statements which are usually entered as a batch mode job script. The Teradata MultiLoad commands perform session control and data handling of the data transfers. The Teradata SQL statements do the actual maintenance functions on the database tables and views.

MultiLoad Phases

Table 5: Phases of a Teradata MultiLoad Operation

Phase	Teradata MultiLoad Operation
Preliminary	1. Parses and validates all of the Teradata MultiLoad commands and Teradata SQL statements in a Teradata MultiLoad job. 2. Establishes sessions and process control with Teradata database. 3. Submits special Teradata SQL requests to Teradata database. 4. Creates and protects temporary work tables and error tables in Teradata database.
DML Transaction	Submits the DML statements specifying the insert, update, and delete tasks to the Teradata database.

Table 5: Phases of a Teradata MultiLoad Operation

Phase	Teradata MultiLoad Operation
Acquisition	1. Imports data from the specified input data source. 2. Evaluates each record according to specified application conditions. 3. Loads the selected records into the worktables in Teradata database. **NOTE:** There is no acquisition phase activity for a Teradata MultiLoad delete task.
Application	1. Acquires locks on the specified target tables and views in Teradata database. 2. For an *import* task, inserts the data from the temporary work tables into the target tables or views in Teradata database. For a *delete* task, deletes the specified rows from the target table in Teradata database. 3. Updates the error tables associated with each Teradata MultiLoad task.
Cleanup	1. Forces an automatic restart/rebuild if an AMP went offline and came back online during the application phase. 2. Releases all locks on the target tables and views. 3. Drops the temporary work tables and all empty error tables from Teradata database. 4. Reports the transaction statistics associated with the import and delete tasks.

MultiLoad Commands

The following generic commands are typically used in a MultiLoad script:

Table 6: MultiLoad Commands

Command	Description
`.LOGTABLE logtname;`	Specifies a restart log table for the Teradata MultiLoad checkpoint information. Teradata MultiLoad uses the information in the restart log table to restart jobs.
`.LOGON database,password;`	establishes a Teradata SQL session with Teradata Database.

`.BEGIN IMPORT MLOAD`
`TABLES tname1`
`WORKTABLES tname2`
`ERRORTABLES tname3`
`tname4`
	`BEGIN MLOAD` and `BEGIN DELETE MLOAD` commands initiate or restart Teradata MultiLoad import or delete tasks.
	`TABLES` specifies the target table or view for an import task.
	`WORKTABLES` specifies the work table for each *`tname1`* table or view. Work tables are special unhashed tables that Teradata MultiLoad uses when executing both import and delete tasks.
	`ERRORTABLES` specifies the fallback error table for each *`tname1`* table or view that receives information about errors detected during the **acquisition** phase of the Teradata MultiLoad import task.
	A second `ERRORTABLES` table is used to specify the fallback error table for each *`tname1`* table or view that receives information about errors detected during the **application** phase of the Teradata MultiLoad import task.

Table 6: MultiLoad Commands (Continued)

Command	Description
.LAYOUT *layoutname* .FIELD	LAYOUT, used with an immediately following sequence of FIELD, FILLER, and TABLE commands, specifies the layout of the input data records. FIELD specifies a field of the input record to be sent to Teradata database.
.DML LABEL	Defines a label and error treatment options for a following group of DML statements.
INSERT INTO *TNAME* (*CNAME*) VALUES (:*FIELDNAME*)	INSERT INTO ... VALUES is a Teradata SQL statement that adds new rows to a table or view.
.IMPORT INFILE *FILE_NAME* FORMAT LAYOUT APPLY *DML_LABEL*	IMPORT specifies a source for data input, for example, a file. FORMAT specifies the format of data in file *FILE_NAME*, for example, FASTLOAD, VARTEXT, UNFORMATTED, TEXT, BINARY, FORMAT. LAYOUT specifies the layout name used in above .LAYOUT command APPLY specifies the DML label name *DML_LABEL* to use.
.END MLOAD	END MLOAD must be the last command of a Teradata MultiLoad task, signifying the end of the task script and initiating task processing by the Teradata database.
.LOGOFF	LOGOFF disconnects all active sessions and terminates Teradata MultiLoad on the client system.

NOTE: For full details on Teradata MultiLoad utility commands see the *Teradata MultiLoad Reference* on https://docs.teradata.com.

Example MultiLoad Script

The following code example shows VPTD integration in a MultiLoad script:

```
.LOGTABLE vormetric.EMP_ENC_log;
.LOGON vormetric,Ssl12345#;
 DATABASE vormetric;


DROP TABLE EMP_ENC;
DROP TABLE UV_EMP_ENC;
DROP TABLE ET_EMP_ENC;
DROP TABLE WT_EMP_ENC;

CREATE MULTISET TABLE EMP_ENC(
ID INTEGER,
NAME VARCHAR(100),
DEPT VARCHAR(100),
NAME_1 VARBYTE(100)
)
UNIQUE PRIMARY INDEX(ID);


.BEGIN IMPORT MLOAD
        TABLES         EMP_ENC
        WORKTABLES     WT_EMP_ENC
        ERRORTABLES    ET_EMP_ENC
                       UV_EMP_ENC;

.LAYOUT INPUTLAYOUT;
    .FIELD in_ID * VARCHAR(50);
    .FIELD in_NAME * VARCHAR(100);
    .FIELD in_DEPT * VARCHAR(100);

.DML LABEL INSERTS IGNORE DUPLICATE INSERT ROWS;
```

```
INSERT INTO EMP_ENC(ID , NAME , DEPT , NAME_1) VALUES
(:in_ID,:in_NAME,:in_DEPT,vormetric.encrypt_cbc(:in_NAME,'encrypt_ccnum'));


.IMPORT INFILE emp_data.txt
       FORMAT VARTEXT ','
       LAYOUT INPUTLAYOUT
APPLY INSERTS;

.END MLOAD;

.LOGOFF;
```

The following shows the contents of the input file `emp_data.txt` used in the example above:

```
1000,Emp_1,development
1001,EMP_2,development
1002,EMP_3,automation
1003,EMP_4,automation
```

Running a MultiLoad Job

Use the following command to invoke Teradata MultiLoad, and run the job script, in this case named `insert.mload`:

```
mload < insert.mload
```

Using UDFs with MLOAD INSERT

encrypt_cbc()

Consider the following encrypt_cbc() UDF declaration:

```
function encrypt_cbc (inputString varchar(8192) CHARACTER SET UNICODE,\
inputKeyname varchar(256)) returns varbyte(16400);
```

In this declaration, encrypt_cbc() takes an input parameter of type VARCHAR having max 8192 characters, and returns encrypted data of type VARBYTE.

In the preceding example (under "Example MultiLoad Script" on page 78), note that in_NAME is set as an input field of type VARCHAR, and is passed to encrypt_cbc(), which satisfies the requirement that inputs match.

The result of the above encrypt_cbc() UDF is the stopring into the NAME_1 column of type VARBYTE, which also satisfies the requirement that outputs match.

You must pass the VARCHAR string as an input parameter, and store the result of the encrypt_cbc() UDF to a column of type VARBYTE. If these requirements are not satisfied, you can get the following errors:

- **UTY0805 RDBMS failure, 9881: Function 'encrypt_cbc' called with an invalid number of type of parameters.**
  This indicates that the input field is not of type VARCHAR.
- **UTY0805 RDBMS failure, 3532: conversion between BYTE data and other types is illegal**
  This indicates you are storing output to a column that is not of type VARBYTE.

encrypt_fpe() and encrypt_ff1()

Consider the following declarations of encrypt_fpe() and encrypt_ff1() UDFs:

```
function encrypt_fpe (inputString varchar(8192) CHARACTER SET UNICODE,\
inputKeyname varchar(256)) returns varchar(8192) CHARACTER SET UNICODE
```

```
function encrypt_ff1 (inputString varchar(8192) CHARACTER SET UNICODE,\
inputKeyname varchar(256)) returns varchar(8192) CHARACTER SET UNICODE
```

The inputs and output are all type VARCHAR.

When using UDFs in an INSERT command of a MultiLoad script, you must provide an input parameter of the same type as specified in the function declaration. You must also store output parameters to a column of that specific type.

Verifying Import Tasks

Use the following BTEQ commands to verify the Teradata MultiLoad import task by selecting newly imported data from the above table:

```
bteq
.LOGON userID,password;
.WIDTH 200
SELECT * FROM EMP_ENC;
.QUIT
```

Decrypting and Checking Data

Use the following BTEQ commands to decrypt and check the data of the original and decrypted columns:

```
bteq
.LOGON userID,password;
SELECT NAME_1, vormetric.decrypt_cbc(NAME_1,'encrypt_ccnum')FROM EMP_ENC;
.quit
```

About FastExport

FastExport is a command-driven utility that uses multiple sessions to quickly transfer large amounts of data from tables and views of a Teradata database into a client-based application or a flat file. You can generate a MultiLoad script from a FastExport job.

When Teradata FastExport is invoked, the utility executes the FastExport commands and Teradata SQL statements in the FastExport job script. These direct FastExport to:

1. Log on to Teradata database for a specified number of sessions, using username, password, and tdpid/acctid information.
2. Retrieve the specified data from Teradata database, in accordance with the format and selection specifications.
3. Export the data to the specified file, or to an OUTMOD routine on a client system.
4. Log off the Teradata database.

Examples

This section describes the following examples:

- “Simple Export/Import” on page 82 - Export table data from Teradata and upload this exported data into a new table.
- “Export with MODE and FORMAT Specifications” on page 86 - Export table data from Teradata using "Mode = INDICATOR and FORMAT = Text," and upload exported data into a new table.
- “Export Encrypted Data after Decryption” on page 88 - Export encrypted data after decrypting it and upload exported data into a new table.

Simple Export/Import

The following procedure shows how to export table data from Teradata and then upload the exported data into a new table.

1. Create a table in Teradata using the CREATE TABLE command, and add data to this table:

```
CREATE MULTISET TABLE EMP_ENC(
ID INTEGER,
NAME VARCHAR(100),
DEPT VARCHAR(100)
)
UNIQUE PRIMARY INDEX(ID);
```

2. Create a FastExport script named `fexport_mload.fexp` that exports data from the above created table using the FastExport Teradata utility. Specify in this script the MLSCRIPT option to generate a MultLoad script named `emp_data_fexp_mload.mload`.

```
.LOGTABLE VORMETRIC.EMP_LOG;
.LOGON userID,password;
database vormetric;

.begin export sessions 12;

.export outfile "emp_data_fexp_mload.txt" MODE RECORD FORMAT FASTLOAD
MLSCRIPT "emp_data_fexp_mload.mload";

select CAST (ID AS CHAR(5)),
    CAST (NAME AS CHAR(10)),
    CAST (DEPT AS CHAR(100))
 from EMP_ENC;

.end export;

.logoff;
```

Use the following command to run the FastExport script:

```
fexp < fexport_mload.fexp
```

3. Modify the generated MultiLoad script (`emp_data_fexp_mload.mload`), and load the exported data into new table by the encrypting column.

```
/*   Date of extract:         THU JAN 09, 2020 */
/*   Time of extract:         01:33:59         */

/*   Total records extracted for select 1 = 4 */
/*      Output record length for select 1 = 115 fixed */


.LOGTABLE vormetric.EMP_ENC_DEST_log;

.LOGON userID,password;

DROP TABLE EMP_ENC_DEST;
DROP TABLE WT_EMP_ENC_DEST;
DROP TABLE ET_EMP_ENC_DEST;
DROP TABLE UV_EMP_ENC_DEST;

CREATE MULTISET TABLE EMP_ENC_DEST(
ID INTEGER,
NAME VARCHAR(100),
DEPT VARCHAR(100)
)
UNIQUE PRIMARY INDEX(ID);

ALTER TABLE EMP_ENC_DEST ADD NAME_1 VARBYTE(100);

.SET DBASE_TARGETTABLE TO 'VORMETRIC';
.SET DBASE_WORKTABLE   TO 'VORMETRIC';
.SET DBASE_ETTABLE     TO 'VORMETRIC';
.SET DBASE_UVTABLE     TO 'VORMETRIC';
.SET TARGETTABLE       TO 'EMP_ENC_DEST';

.BEGIN IMPORT MLOAD
       TABLES &DBASE_TARGETTABLE..&TARGETTABLE
   WORKTABLES &DBASE_WORKTABLE..WT_&TARGETTABLE
  ERRORTABLES &DBASE_ETTABLE..ET_&TARGETTABLE
              &DBASE_UVTABLE..UV_&TARGETTABLE;
```

```
.LAYOUT DATAIN_LAYOUT;
.FIELD ID 1 CHAR(5);
.FIELD NAME 6 CHAR(10);
.FIELD DEPT 16 CHAR(100);

.DML LABEL INSERT_DML;
INSERT INTO &DBASE_TARGETTABLE..&TARGETTABLE (
 ID = :ID
,NAME = :NAME
,DEPT = :DEPT
,NAME_1 = vormetric.encrypt_cbc(:NAME,'encrypt_ccnum')
);

.IMPORT INFILE emp_data_fexp_mload.txt
  FORMAT FASTLOAD
  LAYOUT DATAIN_LAYOUT
  APPLY INSERT_DML;

.END MLOAD;

ALTER TABLE EMP_ENC_DEST
DROP NAME,
RENAME NAME_1 to NAME ;

.LOGOFF &SYSRC;
```

After running the modified MultiLoad script, the table data look like this:

```
 BTEQ -- Enter your SQL request or BTEQ command
select * from EMP_ENC_DEST;

select * from EMP_ENC_DEST;

*** Query completed. 4 rows found. 3 columns returned.
*** Total elapsed time was 1 second.

   ID  DEPT            NAME
-----  ------------  --------------------------------------------------
 1002  automation    4C49D2573D8F97A081201DA63J8DJ762FF893D376D788A956
 1000  development   2E0753DE43AI4B1156D1IEBE5E78DD4I93DD96F1CA6D3D5CD
 1003  automation    5612DCF7F4496D8DB7338DA90E66DE99240D598F5210FEDE8
 1001  development   39062A56B10D1519986BBD89D833F6DD10B67DA048C846DED
```

Export with MODE and FORMAT Specifications

The following procedure shows how to export table data from TeraData using "Mode = INDICATOR and FORMAT = Text," and then upload the exported data into a new table.

1. Export table data into a file in TEXT format, and using MODE INDICATOR in the FastExport script.

```
.LOGTABLE VORMETRIC.EMP_LOG;
.LOGON userID,password;
database vormetric;

.begin export sessions 12;

.export outfile "emp_data_fexp_mload_indicator_text.txt" MODE INDICATOR
FORMAT TEXT MLSCRIPT "emp_data_fexp_mload_indicator_text.mload";

select CAST (ID AS CHAR(5)),
    CAST (NAME AS CHAR(10)),
    CAST (DEPT AS CHAR(100))
 from EMP_ENC;

.end export;

.logoff;
```

2. Modify the generated MultiLoad script `emp_data_fexp_mload_indicator_text.mload` and upload the data to a new column by calling VPTD UDF.

```
/*   Total records extracted for select 1 = 4 */
/*      Output record length for select 1 = 115 fixed */

.LOGTABLE EMP_ENC_INDICATOR_FASTLOAD_log;

.LOGON userID,password;

CREATE MULTISET TABLE EMP_ENC_INDICATOR_FASTLOAD(
ID INTEGER,
NAME VARCHAR(100),
DEPT VARCHAR(100)
/*NAME_1 VARBYTE(100)*/
)
UNIQUE PRIMARY INDEX(ID);
```

```
ALTER TABLE EMP_ENC_INDICATOR_FASTLOAD ADD NAME_1 VARBYTE(100);

.SET DBASE_TARGETTABLE TO 'VORMETRIC';
.SET DBASE_WORKTABLE   TO 'VORMETRIC';
.SET DBASE_ETTABLE     TO 'VORMETRIC';
.SET DBASE_UVTABLE     TO 'VORMETRIC';
.SET TARGETTABLE       TO 'EMP_ENC_INDICATOR_FASTLOAD';

.BEGIN IMPORT MLOAD
       TABLES &DBASE_TARGETTABLE..&TARGETTABLE
   WORKTABLES &DBASE_WORKTABLE..WT_&TARGETTABLE
  ERRORTABLES &DBASE_ETTABLE..ET_&TARGETTABLE
              &DBASE_UVTABLE..UV_&TARGETTABLE;

.LAYOUT DATAIN_LAYOUT INDICATORS;
.FIELD ID 1 CHAR(5);
.FIELD NAME 6 CHAR(10);
.FIELD DEPT 16 CHAR(100);

.DML LABEL INSERT_DML;
INSERT INTO &DBASE_TARGETTABLE..&TARGETTABLE (
 ID = :ID
,NAME = :NAME
,DEPT = :DEPT
,NAME_1 = vormetric.encrypt_cbc(:NAME,'encrypt_ccnum')
);

.IMPORT INFILE emp_data_fexp_mload_indicator_fastload.txt
  FORMAT TEXT
  LAYOUT DATAIN_LAYOUT
  APPLY INSERT_DML;

.END MLOAD;

ALTER TABLE EMP_ENC_INDICATOR_FASTLOAD
DROP NAME,
RENAME NAME_1 to NAME;

.LOGOFF &SYSRC;
```

Export Encrypted Data after Decryption

The following procedure shows how to export encrypted data after decrypting it, and upload the exported data into a new table.

Consider the following table `EMP_ENC_INDICATOR_FASTLOAD` containing encrypted data:

```
 BTEQ -- Enter your SQL request or BTEQ command
select * from EMP_ENC_INDICATOR_FASTLOAD;

select * from EMP_ENC_INDICATOR_FASTLOAD;

*** Query completed. 4 rows found. 3 columns returned.
*** Total elapsed time was 1 second.

   ID  DEPT            NAME
-----  ------------  --------------------------------------------------
 1002  automation    4C49D2573D8F97A081201DA63J8DJ762FF893D376D788A956
 1000  development   2E0753DE43AI4B1156D1IEBE5E78DD4I93DD96F1CA6D3D5CD
 1003  automation    5612DCF7F4496D8DB7338DA90E66DE99240D598F5210FEDE8
 1001  development   39062A56B10D1519986BBD89D833F6DD10B67DA048C846DED

 BTEQ -- Enter your SQL request or BTEQ command
```

1. Export data in decrypted format using a FastExport script:

```
.LOGTABLE VORMETRIC.EMP_LOG;
.LOGON userID,password;
database vormetric;

.begin export sessions 12;

.export outfile "emp_data_fexp_mload.txt" MODE INDICATOR FORMAT
FASTLOAD MLSCRIPT "emp_data_fexp_mload.mload";

select CAST (ID AS CHAR(5)),
    CAST (DEPT AS CHAR(15)),
    vormetric.decrypt_cbc(NAME,'encrypt_ccnum')
 from EMP_ENC_INDICATOR_FASTLOAD;

.end export;

.logoff;
```

2. Upload the decrypted data to a new table using the generated MultiLoad script `emp_data_fexp_mload.mload`.

```
.LOGTABLE LOGTABLE053439;

.LOGON userID,password;

CREATE MULTISET TABLE EMP_ENC_TEST(
ID INTEGER,
NAME VARCHAR(100),
DEPT VARCHAR(100)
)
UNIQUE PRIMARY INDEX(ID);


.SET DBASE_TARGETTABLE TO 'VORMETRIC';
.SET DBASE_WORKTABLE   TO 'VORMETRIC';
.SET DBASE_ETTABLE     TO 'VORMETRIC';
.SET DBASE_UVTABLE     TO 'VORMETRIC';
.SET TARGETTABLE       TO 'EMP_ENC_TEST';

.BEGIN IMPORT MLOAD
       TABLES &DBASE_TARGETTABLE..&TARGETTABLE
   WORKTABLES &DBASE_WORKTABLE..WT_&TARGETTABLE
  ERRORTABLES &DBASE_ETTABLE..ET_&TARGETTABLE
              &DBASE_UVTABLE..UV_&TARGETTABLE;

.LAYOUT DATAIN_LAYOUT INDICATORS;
.FIELD ID 1 CHAR(5);
.FIELD DEPT 6 CHAR(15);
.FIELD  NAME 21 VARCHAR(8192);

.DML LABEL INSERT_DML;
INSERT INTO &DBASE_TARGETTABLE..&TARGETTABLE (
 ID = :ID
,DEPT = :DEPT
,NAME = :NAME
);
```

```
.IMPORT INFILE emp_data_fexp_mload.txt
  FORMAT FASTLOAD
  LAYOUT DATAIN_LAYOUT
  APPLY INSERT_DML;

.END MLOAD;

.LOGOFF &SYSRC;
```

3. The table EMP_ENC_TEST now contains the following decrypted data:

```
 BTEQ -- Enter your SQL request or BTEQ command
select * from EMP_ENC_TEST;

select * from EMP_ENC_TEST;

*** Query completed. 4 rows found. 3 columns returned.
*** Total elapsed time was 1 second.

   ID  NAME                                DEPT
-----  ----------------------------------  ----------------------
 1002  EMP_3                               automation
 1000  EMP_1                               development
 1003  EMP_4                               automation
 1001  EMP_2                               development

 BTEQ -- Enter your SQL request or BTEQ command
```

VPTD Ongoing Operations

5

This chapter describes ongoing operations for Vormetric Protection for Teradata Database. This chapter contains the following sections:

Log Messages

See `/var/log/messages` for error messages originating from the Vormetric Local Cryptoserver Daemon.

See `/var/log/vormetric/pkcs11_root.log` for information messages originating when encrypt or decrypt operations are performed by the local Cryptoserver Daemon on behalf of the UDFs.

See `/var/log/vormetric/vptd_icapi.log` for information messages originating from the ICAPI library when encrypt or decrypt operations are performed by the local Cryptoserver Daemon on behalf of the UDFs. This log file is generated only when VPTD is configured with a KeySecure server.

NOTE: During long-running queries, if the log level is set to INFO or DEBUG, the volume of log data sometimes overwhelms the system, and the process might appear to hang.

Always set the log level to ERROR before trying any long-running query.

To set log level to error:

- Open the `/etc/vormetric/vormetric_local_crypto_server.conf` file.

- Change the loglevel key with loglevel error.

- Restart the crypto server with:

```
# /etc/init.d/vormetric_local_crypto_server restart
```

Troubleshooting

During ongoing operation of Vormetric Protection for Teradata Database, the following issues may arise if the appropriate steps are not taken to avoid them. Be aware of these considerations and take the recommended steps to ensure smooth operation.

Be sure Teradata users can access /tmp/vormetric

The Vormetric Teradata UDFs and the local Vormetric Cryptoserver communicate through the named socket `/tmp/vormetric`. The UDFs run in the Teradata context, so this socket must have access permissions set to 666 to allow access to the Teradata user. If only the root user has permission to access `/tmp/vormetric`, the UDF cannot communicate with the Cryptoserver and the UDF will fail.

Cache key on host when turning off udf_aes

If using the `encrypt_fpe()` and `decrypt_fpe()` UDFs with the `udf_aes` option off, be sure the encryption key created on the Vormetric DSM is cached on the host.

Otherwise, the UDF generates the following error:

```
error '500 - C_EncryptInit failed'
```

Set width of BTEQ session to avoid truncated UDF output

If you are using `encrypt_cbc()` to encrypt long strings, use the following steps to ensure the output of the encryption UDF is not truncated. If the BTEQ session width is too narrow, characters can be lost from the end of the returned string.

1. Invoke BTEQ with the following command:
   ```
   # bteq -e UTF8 -c UTF16
   ```
2. At the BTEQ prompt, run the following command. Instead of 1000, substitute any value that ensures the width is sufficient:
   ```
   # .set width 1000
   ```
3. Run the following command:
   ```
   # .set session charset "utf8"
   ```

Flush the cache to remove old profiles

When you change the files `profiles.conf`, `mask.conf`, or `vormetric_local_crypto_server.conf`, the updates are not necessarily recognized

immediately. Information from an older configuration file could be cached. To be sure the most up-to-date information is available, flush the BTEQ cache using the following steps.

Caution: The database is restarted during this procedure. If any critical operations are underway, wait until they have finished.

After changes are made to either `profiles.conf` or `vormetric_local_crypto_server.conf`:

1. Restart the Cryptoserver.
2. Run the following command:
   ```
   # tpareset -f flushing
   ```
3. Run the following command:
   ```
   # pdestate -a
   ```
4. Look for output similar to the following. If this output is not seen, repeat the `pdestate -a` command.
   ```
   PDE state is RUN/STARTED.
   DBS state is 4: Logons are enabled - Users are logged on
   ```

Properly escape characters in input strings

If an improperly formatted string is passed as input to BTEQ, such as when calling the UDFs `encrypt_fpe` or `encrypt_cbc`, the BTEQ session stops and waits indefinitely. The input string must adhere to BTEQ requirements, such as escaping special characters. For example, if the input string contains an apostrophe, use a double apostrophe: 'It''s a beautiful day'. For more information about the requirements for input strings, see Teradata BTEQ documentation.

FF1 license considerations

After modifying the FF1 license, or after a successful upgrade, you must restart the cryptographic server and flush the UDF cache using the **tpareset** command to clear the old license information.

Warning! Failure to complete this step causes the application to fail.

Change the log level if queries hang

During long-running queries, if the log level is set to INFO or DEBUG, the volume of log data sometimes overwhelms the system, and the process might appear to hang. Changing the log level can resolve this problem. See the note under “Log Messages” on page 91 for details.

Locking Down Internet-facing Servers Supporting VPTD

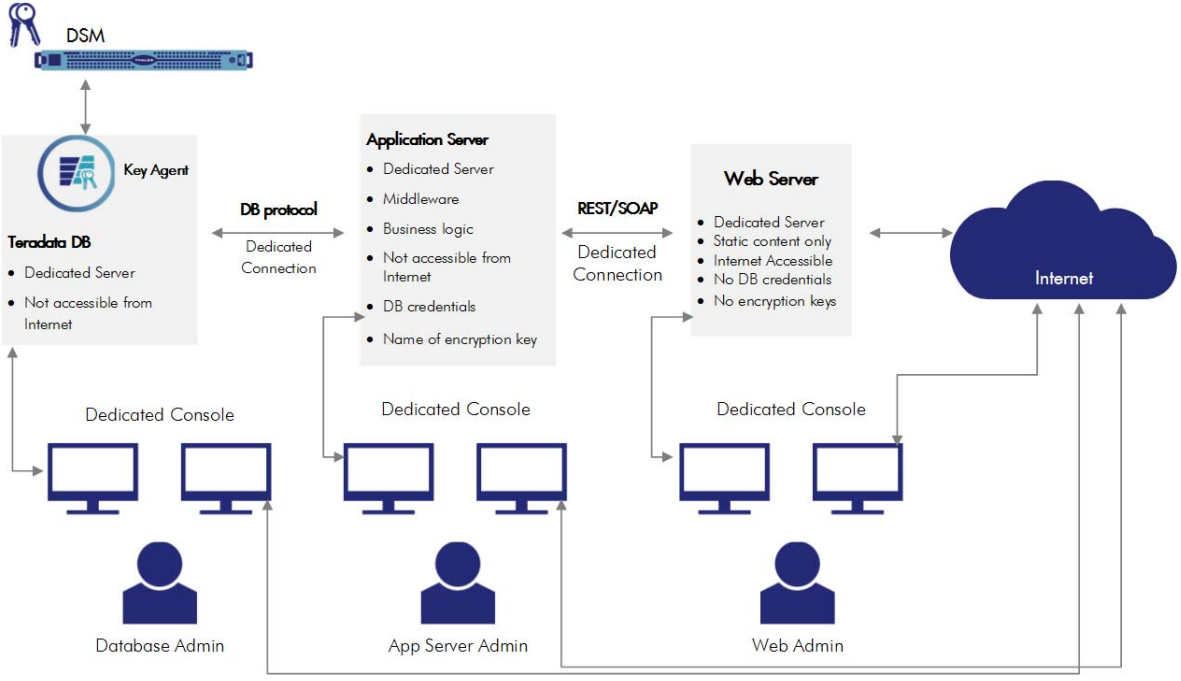
6

This chapter contains the following sections:

Security Tips for Vormetric Protection for Teradata Database

We recommend having at least two dedicated servers—a Web Server and an Application Server—between the Teradata DB and the Internet. This is shown in the following figure.

Figure 6: High Security Deployment for Teradata Deployment



- **Web Server**. The Internet-facing Web Server should be dedicated (a separate machine—virtual ok--with separate address space, IP, firewall, and so on). You must assume the Web Server will be penetrated, and so ensure that it does not contain DB credentials, encryption key names, or any sensitive data. It can contain static content such as static web pages, graphics and video, but it should not contain business logic or have direct access to the Teradata database. It should have such security features as firewalls, but you should not overload it with functionality.

  The Web Server's access to the Teradata DB should be through an Application Server. The Web Server's access to the secure data should come in the form of REST or SOAP API calls to the Application Server, which are fairly traceable, testable, and can be limited in scope.
- **Application Server** should be dedicated and should not be accessible from the Internet. It will have Teradata DB credentials and the name of encryption keys, but not the encryption keys themselves. It will contain the organization's business logic, and when the Application Server receives REST/SOAP requests from the Web Server, it will retrieve the required data from the Teradata DB and send it to the Web Server which will send it along to the requester.
- **Teradata DB** should be dedicated and inaccessible from the Internet. It should pass data only to the Application Server with the appropriate credentials. It should contain the Vormetric Application Encryption (VAE) agent, which is connected to the DSM.

To support separation of duties, each of the above servers should be administered by a different administrator, a Web Administrator, Application Server Administrator, and Database Administrator. The Web Administrator will not have access to the database, nor access to the key name. The Database Administrator has access to the encrypted data, but lacks the key name to decrypt the data. The Application Server Administrator could decompile the middleware and extract the database credentials and the key name(s), but this is not a straight-forward process as long as the application logic is compiled.

Each administrator should perform their administrative duties on their respective servers using a UNIX/Linux or OSX machine, not a Windows machine, which is more vulnerable to hacking. Furthermore, this administrator machine should have an air gap (physical and electronic isolation) from other computers.

The connection from these server administration machines to the dedicated servers should use the following security enhancements:

- Disable unused services such as printer daemons, mail servers and so on.
- Restrict the IP addresses that can log in. For example, only allow connections from corporate headquarters. This can be configured in the SSH configuration and settings.
- Change the SSH port from the default 22 to a random port number (example: 50712). In `/etc/ssh/sshd_config` change port 22 to port 50712
- Disallow SSH password login and use a key pair login instead (see below).

To Disallow SSH Password Login and Use a Key Pair Login

To set up SSH login to disallow passwords and use only key-pair login, first change the following two parameters in the `/etc/ssh/sshd_config` file:

```
PasswordAuthentication no
ChallengeResponseAuthentication no
```

The SSH server will only allow login for users with an SSH key which has been added to `~/.ssh/authorized_keys` file.

Generate user-specific SSH keys by using `ssh-keygen`, which yields a private key file and a public key file (example: `id_rsa` and `id_rsa.pub`).

The private key file must remain in the workstation you use to access your production server. The public key file, however, needs to be appended to the `authorized_keys` file on the server.

Specify a password during key file generation. We strongly recommend password-protecting your private key. If you don't password-protect your private key, anyone with access to your computer conceivably can SSH (without being prompted for a password) to your account on any remote system that has the corresponding public key. You may use an SSH key without a password for automated administration tasks such as administrative shell scripts. However, restrict the scope of these keys by prefixing them with the particular command for which they will be used. For example:

```
command="rsync --server --sender -vlHogDtprxe.iLsf --numeric-ids .
/srv" ssh-rsa AAAA ...
```

These SSH keys used for automation tasks should not use root access. If root access is required, use `sudo`.

You may consider denying root logins by setting the `PermitRootLogin` to `no` in the `sshd_config` file, or allow them only for logins which are tied to a particular command using the `forced-commands-only` option.

7

Uninstalling VPTD

This chapter contains the following sections:

Uninstall Vormetric Protection for Teradata Database

When trying to uninstall VPTD on a node, the uninstaller detects the presence of the cluster and will try to uninstall VPTD from the other nodes that contain the same version of VPTD. If there is only a single node, then it uninstalls VPTD from that node.

The uninstall script also ensures that the crypto server daemon is not running on any single node before uninstalling.

NOTE: Do not call the uninstaller from within `/opt/vormetric`. You must uninstall from outside of this location.

1. Remove Teradata UDFs by default in a specific database named "vormetric". Either drop the database or remove the UDFs (user-defined functions) by running the following binary file:

```
# cd /opt/vormetric/DataSecurityExpert/agent/pkcs11/teradata/udfs
# ./drop_udfs.bteq
Enter password for vormetric user:
Confirm password for vormetric user:
#
```

2. Uninstall the VAE executable on Teradata:

```
rpm -qa | grep vae
vae-td14.10-6.3.0-94
rpm -e vae-td14.10-6.3.0-94
```

3. Uninstall files in `/opt/vormetric/DataSecurityExpert/agent` using `/opt/vormetric/DataSecurityExpert/agent/key/bin/uninstall`. These include:
   - Vormetric key agent

- `/opt/vormetric/DataSecurityExpert/agent/pkcs11/teradata/udfs/install_udfs.bteq.sample`
- `/opt/vormetric/DataSecurityExpert/agent/pkcs11/teradata/udfs/<renamed bteq.sample file>`
- `/opt/vormetric/DataSecurityExpert/agent/pkcs11/Teradata/bin/vormetric_local_crypto_server`

4. Uninstall any files that are present in `/etc/vormetric/` with `rm -f`.
5. Remove the `init.d` script to start the local crypto server:
   - Delete `/etc/init.d/vormetric_local_crypto_server`

Automated Uninstall over a Teradata Cluster

The intent of this feature is to detect a cluster and ensure that the uninstall is performed on all nodes in the cluster.

NOTE: You must ensure that the crypto server daemon is not running on any single node before the uninstall begins.

When trying to uninstall VPTD on a node, the uninstaller detects the presence of the cluster and tries to uninstall the same VPTD on the other nodes. If there is a single node, then it also uninstalls the VPTD from that node as well.

The uninstaller is located in the following directory:

`/opt/vormetric/DataSecurityExpert/agent/pkcs11/teradata/bin`

Warning! The uninstaller cannot be called from `/opt/vormetric` or its sub-directories. You can call the uninstaller from any other directory.

To uninstall VPTD:

1. Remove Teradata UDFs by default in a specific database named “vormetric”. Either drop the database or remove the UDFs (user-defined functions) by running the following binary file:

```
# cd /opt/vormetric/DataSecurityExpert/agent/pkcs11/teradata/udfs
# ./drop_udfs.bteq
Enter password for vormetric user:
Confirm password for vormetric user:
#
```

2. Enter:

```
# /opt/vormetric/DataSecurityExpert/agent/pkcs11/teradata/bin/vptd_uninstall
```