Protegrity Big Data Protector Guide

Release 6.6.5

Copyright

Copyright © 2004-2017 Protegrity Corporation. All rights reserved.

Protegrity products are protected by and subject to patent protections;
Patent:http://www.protegrity.com/patents

Protegrity logo is the trademark of Protegrity Corporation.

NOTICE TO ALL PERSONS RECEIVING THIS DOCUMENT

Some of the product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective owners.

Windows, MS-SQL Server, Internet Explorer and Internet Explorer logo, Active Directory, and Hyper-V are registered trademarks of Microsoft Corporation in the United States and/or other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SCO and SCO UnixWare are registered trademarks of The SCO Group.

Sun, Oracle, Java, and Solaris, and their logos are the trademarks or registered trademarks of Oracle Corporation and/or its affiliates in the United States and other countries.

Teradata and the Teradata logo are the trademarks or registered trademarks of Teradata Corporation or its affiliates in the United States and other countries.

Hadoop or Apache Hadoop, Hadoop elephant logo, HDFS, Hive, Pig, HBase, and Spark are trademarks of Apache Software Foundation.

Cloudera, Impala, and the Cloudera logo are trademarks of Cloudera and its suppliers or licensors.

Hortonworks and the Hortonworks logo are the trademarks of Hortonworks, Inc. in the United States and other countries.

Greenplum is the registered trademark of EMC Corporation in the U.S. and other countries.

Pivotal HD and HAWQ are the registered trademarks of Pivotal, Inc. in the U.S. and other countries.

MapR logo is a registered trademark of MapR Technologies, Inc.

PostgreSQL or Postgres is the copyright of The PostgreSQL Global Development Group and The Regents of the University of California.

IBM and the IBM logo, z/OS, AIX, DB2, Netezza, and BigInsights are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Utimaco Safeware AG is a member of the Sophos Group.

Jaspersoft, the Jaspersoft logo, and JasperServer products are trademarks and/or registered trademarks of Jaspersoft Corporation in the United States and in jurisdictions throughout the world.

Xen, XenServer, and Xen Source are trademarks or registered trademarks of Citrix Systems, Inc. and/or one or more of its subsidiaries, and may be registered in the United States Patent and Trademark Office and in other countries.

VMware, the VMware “boxes” logo and design, Virtual SMP and VMotion are registered trademarks or trademarks of VMware, Inc. in the United States and/or other jurisdictions.

HP is a registered trademark of the Hewlett-Packard Company.

Dell is a registered trademark of Dell Inc.

Novell is a registered trademark of Novell, Inc. in the United States and other countries.

POSIX is a registered trademark of the Institute of Electrical and Electronics Engineers, Inc.

Mozilla and Firefox are registered trademarks of Mozilla foundation.

Chrome is a registered trademark of Google Inc.

Contents

1 Introduction to this Guide

This guide provides information about installing, configuring, and using the Protegrity Big Data Protector (BDP) for Hadoop.

1.1. Sections contained in this Guide

The guide is broadly divided into the following sections:

- Section *1 Introduction to this Guide* defines the purpose and scope for this guide. In addition, it explains how information is organized in this guide.
- Section *2 Overview of the Big Data Protector* provides a general idea of Hadoop and how it has been integrated with the Big Data Protector. In addition, it describes the protection coverage of various Hadoop ecosystem applications, such as MapReduce, Hive and Pig, and information about HDFS File Protection (HDFSFP).
- Section *3 Installing and Uninstalling Big Data Protector* includes information common to all distributions, such as prerequisites for installation, installation procedure and uninstallation of the product from the cluster. In addition, it provides information about the tools and utilities.
- Section *4 Hadoop Application Protector* provides information about Hadoop Application Protector. In addition, it covers information about MapReduce APIs and Hive and Pig UDFs.
- Section *5 HDFS File Protector (HDFSFP)* provides information about the protection of files stored in HDFSFP and the commands supported.
- Section *6 HBase* provides information about the Protegrity HBase protector.
- Section *7 Impala* provides information about the Protegrity Impala protector.
- Section *8 HAWQ* provides information about the Protegrity HAWQ protector.
- Section *9 Spark* provides information about the Protegrity Spark protector. In addition, it provides information about Spark SQL and Spark Scala.
- Section *10 Data Node and Name Node Security with File Protector* provides information about the protection of the Data and Name nodes using the File Protector.
- Section *11 Appendix: Return Codes* provides information about all possible error codes and error descriptions for Big Data Protector.
- Section *12 Appendix: Samples* provides information about sample data protection for MapReduce, Hive, Pig, HBase, Impala, HAWQ, and Spark using Big Data Protector.
- Section *13 Appendix: HDFSFP Demo* provides information about sample data protection for HDFSFP using Big Data Protector.
- Section *14 Appendix: Using Hive with HDFSFP* provides information about using Hive with HDFSFP.
- Section *15 Appendix: Configuring Talend with HDFSFP* provides the procedures for configuring Talend with HDFSFP.
- Section *16 Appendix: Migrating Tokenized Unicode Data from and to a Teradata Database* describes procedures for migrating tokenized Unicode data from and to a Teradata database.

1.2. Protegrity Documentation Suite

The Protegrity Documentation Suite comprises of the following documents:

- *Protegrity Documentation Master Index Release 6.6.5*
- *Protegrity Appliances Overview Release 6.6.5*
- *Protegrity Enterprise Security Administrator Guide Release 6.6.5*
- *Protegrity File Protector Gateway Server User Guide Release 6.6.4*
- *Protegrity Protection Server Guide Release 6.6.5*

- *Protegrity Data Security Platform Feature Guide Release 6.6.5*
- *Protegrity Data Security Platform Licensing Guide Release 6.6*
- *Protegrity Data Security Platform Upgrade Guide Release 6.6.5*
- *Protegrity Reports Guide Release 6.6.5*
- *Protegrity Troubleshooting Guide Release 6.6.5*
- *Protegrity Application Protector Guide Release 6.5 SP2*
- *Protegrity Big Data Protector Guide Release 6.6.5*
- *Protegrity Database Protector Guide Release 6.6.5*
- *Protegrity File Protector Guide Release 6.6.4*
- *Protegrity Protection Enforcements Point Servers Installation Guide Release 6.6.5*
- *Protegrity Protection Methods Reference Release 6.6.5*
- *Protegrity Row Level Protector Guide Release 6.6.5*
- *Protegrity Enterprise Security Administrator Quick Start Guide Release 6.6*
- *Protegrity File Protector Gateway Server Quick Start Guide Release 6.6.2*
- *Protegrity Protection Server Quick Start Guide Release 6.6*

1.5 Glossary

This section includes Protegrity specific terms, products, and abbreviations used in this document.

Name	***Description***
BDP	The Big Data Protector (BDP) is the API for protecting data on platforms such as Hive, Impala and HBase.
ESA	Enterprise Security Administrator (ESA)
DPS roles	The DPS roles relate to the security policy in the ESA and control the access permissions to the Access Keys. For instance, if a user does not have the required DPS role, then the user would not have access to Access Keys.
DPS	Protegrity Data Protection System (DPS) is the entire system where security policies are defined and enforced, including ESA and Protectors.

2 Overview of the Big Data Protector

The Protegrity Big Data Protector for Apache Hadoop is based on the Protegrity Application Protector. The data is split and shared with all the data nodes in the Hadoop cluster. The Big Data Protector is deployed on each of these nodes and the PEP Server, where the protection enforcement policies are shared.

The Protegrity Big Data Protector is scalable and new nodes can be added as required. It is cost effective since massively parallel computing is done on commodity servers, and it is flexible as it can work with data from any number of sources. The Big Data Protector is fault tolerant as the system redirects the work to another node if a node is lost. It can handle all types of data, such as structured and unstructured data, irrespective of their native formats.

The Big Data Protector protects data, which is handled by various Hadoop applications and protects files stored in the cluster. MapReduce, Hive, Pig, HBase, and Impala can use Protegrity protection interfaces to protect data as it is stored or retrieved from the Hadoop cluster. All standard protection techniques offered by Protegrity are applicable to Big Data Protector.

For more information about the available protection options, such as data types, Tokenization or Encryption types, or length preserving and non-preserving tokens, refer to *Protection Methods Reference Guide 6.6.5*.

2.1 Components of Hadoop

The Big Data Protector works on the Hadoop framework as shown in the following figure.

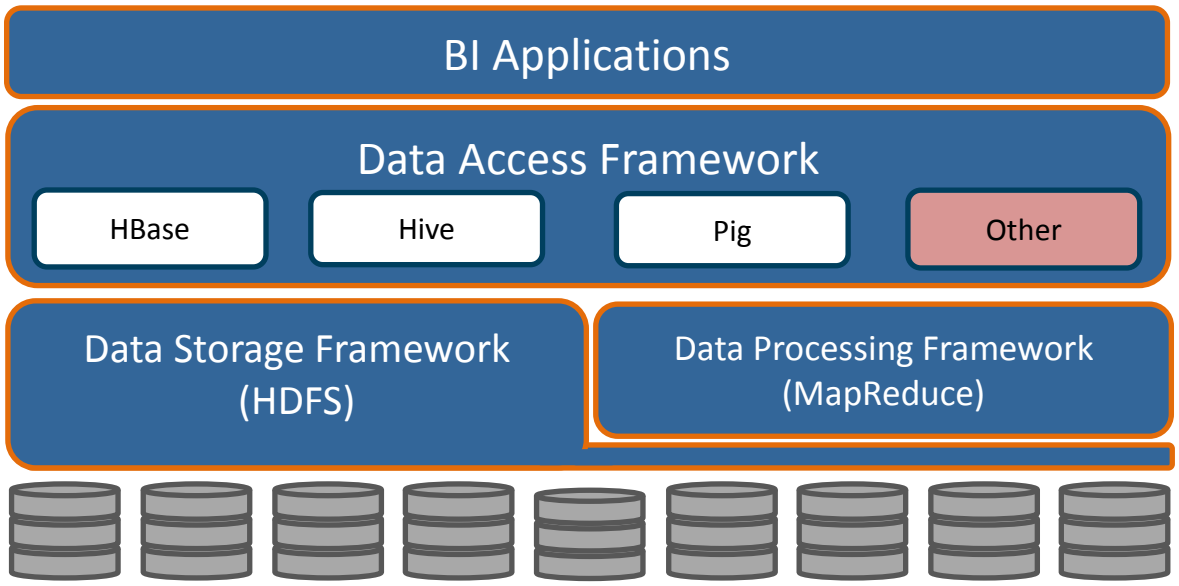


Figure 2-1 Hadoop Components

The illustration of Hadoop components is an example.

Based on requirements, the components of Hadoop might be different.

Hadoop interfaces have been used extensively to develop the Big Data Protector. It is a common deployment practice to utilize Hadoop Distributed File System (HDFS) to store the data, and let MapReduce process the data and store the result back in HDFS.

2.1.1 Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS) spans across all nodes in a Hadoop cluster for data storage. It links together the file systems on many nodes to make them into one big file system. HDFS assumes that nodes will fail, so data is replicated across multiple nodes to achieve reliability.

2.1.2 MapReduce

The MapReduce framework assigns work to every node in large clusters of commodity machines. MapReduce programs are sets of instructions to parse the data, create a map or index, and aggregate the results. Since data is distributed across multiple nodes, MapReduce programs run in parallel, working on smaller sets of data.

A MapReduce job is executed by splitting each job into small Map tasks, and these tasks are executed on the node where a portion of the data is stored. If a node containing the required data is saturated and not able to execute a task, then MapReduce shifts the task to the least busy node by replicating the data to that node. A Reduce task combines results from multiple Map tasks, and store all of them back to the HDFS.

2.1.3 Hive

The Hive framework resides above Hadoop to enable ad hoc queries on the data in Hadoop. Hive supports HiveQL, which is similar to SQL. Hive translates a HiveQL query into a MapReduce program and then sends it to the Hadoop cluster.

2.1.4 Pig

Pig is a high-level platform for creating MapReduce programs used with Hadoop.

2.1.5 HBase

HBase is a column-oriented datastore, meaning it stores data by columns rather than by rows. This makes certain data access patterns much less expensive than with traditional row-oriented relational database systems. The data in HBase is protected transparently using Protegrity HBase coprocessors.

2.1.6 Impala

Impala is an MPP SQL query engine for querying the data stored in a cluster. It provides the flexibility of the SQL format and is capable of running the queries on HDFS in HBase.

The Impala daemon runs on each node in the cluster, reading and writing to data in the files, and accepts queries from the Impala shell command. The following are the core components of Impala:

- Impala daemon (*impalad*) – This component is the Impala daemon which runs on each node in the cluster. It reads and writes the data in the files and accepts queries from the Impala shell command.
- Impala Statestore (*statestored*) – This component checks the health of the Impala daemons on all the nodes contained in the cluster. If a node is unavailable due to any error or failure, then the Impala *statestore* component informs all other nodes about the failed node to ensure that new queries are not sent to the failed node.
- Impala Catalog (*catalogd*) – This component is responsible for communicating any changes in the metadata received from the Impala SQL statements to all the nodes in the cluster.

2.1.7 HAWQ

HAWQ is an MPP database, which uses several Postgres database instances and HDFS storage. The database is distributed across HAWQ segments, which enable it to achieve data and processing parallelism.

Since HAWQ uses the Postgres engine for processing queries, the query language is similar to PostgresSQL. Users connect to the HAWQ Master and interact using SQL statements, similar to the Postgres database.

The following are the core components of HAWQ:

- HAWQ Master Server: Enables users to interact with HAWQ using client programs, such as PSQL or APIs, such as JDBC or ODBC
- Name Node: Enables client applications to locate a file
- HAWQ Segments: Are the units which process the individual data modules simultaneously
- HAWQ Storage: Is HDFS, which stores all the table data
- Interconnect Switch: Is the networking layer of HAWQ, which handles the communication between the segments

2.1.8 Spark

Spark is an execution engine that carries out batch processing of jobs in-memory and handles a wider range of computational workloads. In addition to processing a batch of stored data, Spark is capable of manipulating data in real time.

Spark leverages the physical memory of the Hadoop system and utilizes Resilient Distributed Datasets (RDDs) to store the data in-memory and lowers latency, if the data fits in the memory size. The data is saved on the hard drive only if required.

2.2 Features of Protegrity Big Data Protector

The Protegrity Big Data Protector (Big Data Protector) uses patent-pending vaultless tokenization and central policy control for access management and secures sensitive data at rest in the following areas:

- Data in HDFS
- Data used during MapReduce, Hive and Pig processing, and with HBase, Impala, HAWQ, and Spark
- Data traversing enterprise data systems

The data is protected from internal and external threats, and users and business processes can continue to utilize the secured data.

Data protection may be by encryption or tokenization. In tokenization, data is converted to similar looking inert data known as tokens where the data format and type can be preserved. These tokens can be detokenized back to the original values when it is required.

Protegrity secures files with volume encryption and also protects data inside files using tokenization and strong encryption protection methods. Depending on the user access rights and the policies set using Policy management in ESA, this data is unprotected.

The Protegrity Hadoop Big Data Protector provides the following features:

- Provides fine grained field-level protection within the MapReduce, Hive, Pig, HBase, and Spark frameworks.

- Provides directory and file level protection (encryption).
- Retains distributed processing capability as field-level protection is applied to the data.
- Protects data in the Hadoop cluster using role-based administration with a centralized security policy.
- Provides logging and viewing data access activities and real-time alerts with a centralized monitoring system.
- Ensures minimal overhead for processing secured data, with minimal consumption of resources, threads and processes, and network bandwidth.
- Performs file and volume encryption including the protection of files on the local file system of Hadoop nodes.
- Provides transparent data protection and row level filtering based on the user profile with Protegrity HBase protectors.
- Transparently protects files processed by MapReduce and Hive in HDFS using HDFSFP.

The following figure illustrates the various components in an Enterprise Hadoop ecosystem.

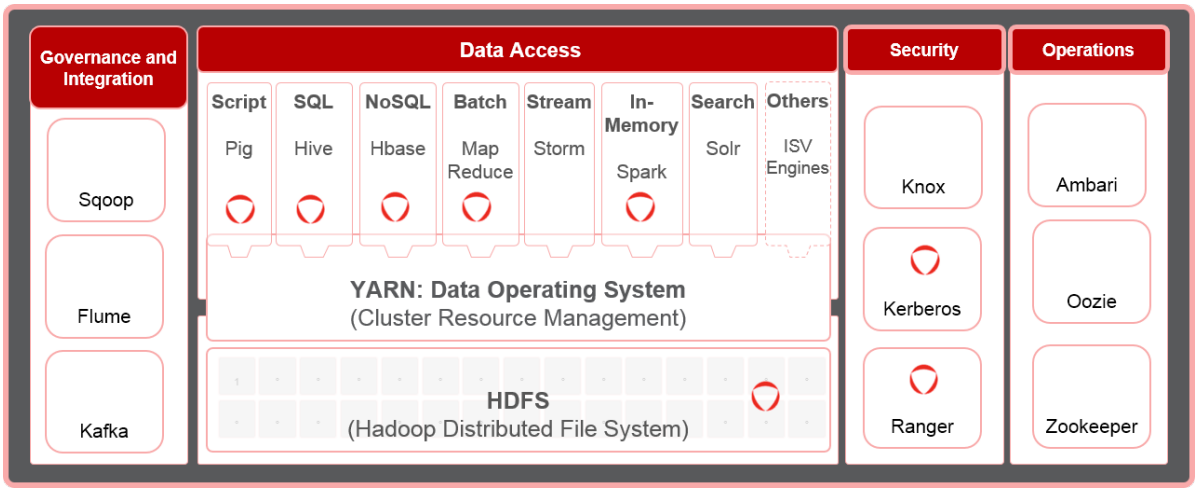


Figure 2-2 Enterprise Hadoop Components

Currently, Protegrity supports MapReduce, Hive, Pig, and HBase which utilize HDFS as the data storage layer. The following points can be referred to as general guidelines:

- Sqoop: Sqoop can be used for ingestion into HDFSFP protected zone (For Hortonworks, Cloudera and Pivotal HD).
- Beeline, Beeswax, and Hue on Cloudera: Beeline, Beeswax, and Hue are certified with Hive protector and Hive with HDFSFP integrations.
- Beeline, Beeswax, and Hue on Hortonworks & Pivotal HD: Beeline, Beeswax, and Hue are certified with Hive protector and Hive with HDFSFP integrations.
- Ranger (Hortonworks): Ranger is certified to work with the Hive protector and Hive with HDFSFP integrations only.
- Sentry (Cloudera): Sentry is certified with Hive protector, Hive with HDFSFP integrations, and Impala protector only.
- MapReduce and HDFSFP integration is certified with *TEXTFILE* format only.

- Hive and HDFSFP integration is certified with *TEXTFILE*, *RCFile*, and *SEQUENCEFILE* formats only.
- Pig and HDFSFP integration is certified with *TEXTFILE* format only.

We neither support nor have certified other components in the Hadoop stack. We strongly recommend consulting Protegrity, before using any unsupported components from the Hadoop ecosystem with our products.

2.3 Using Protegrity Data Security Platform with Hadoop

To protect data, the components of the Protegrity Data Security Platform are integrated into the Hadoop cluster as shown in the following figure.

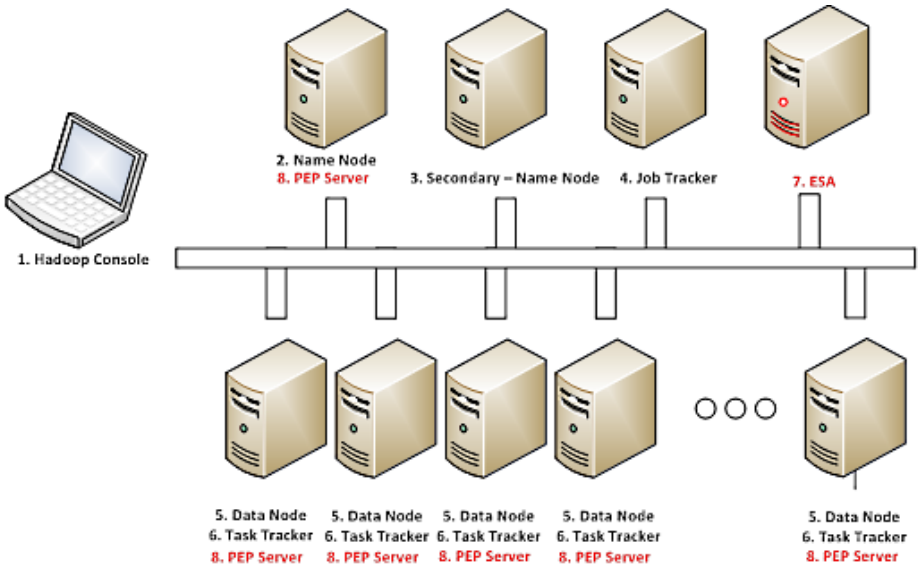


Figure 2-3 Protegrity Data Security Platform with Hadoop

The Enterprise Security Administrator (ESA) is a soft appliance that needs to be pre-installed on a separate server, which is used to create and manage policies.

The following figure illustrates the inbound and outbound ports that need to be allowed on the network for communication between the ESA and the Big Data Protector nodes in a Hadoop cluster.

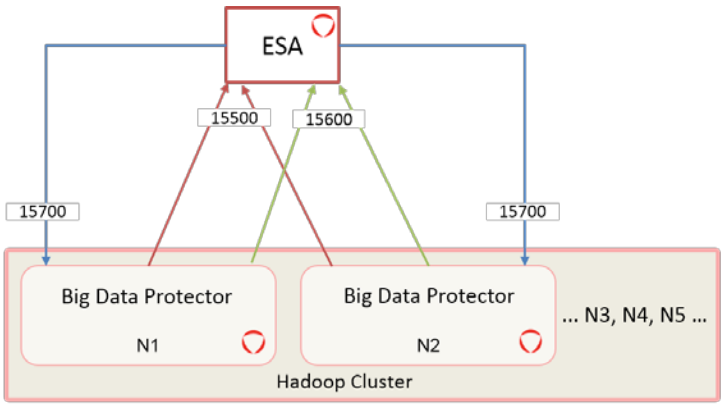


Figure 2-4 Inbound and Outbound Ports between the ESA and Big Data Protector Nodes

For more information about installing the ESA, and creating and managing policies, refer *Protegrity Enterprise Security Administrator Guide Release 6.6.5*.

To achieve a parallel nature for the system, a PEP Server is installed on every data node. It is synchronized with the connection properties of ESA.

Each task runs on a node under the same Hadoop user. Every user has a policy deployed for running their jobs on this system. Hadoop manages the accounts and users. You can get the Hadoop user information from the actual job configuration.

HDFS implements a permission model for files and directories, based on the Portable Operating System Interface (POSIX) for Unix model. Each file and directory is associated with an owner and a group. Depending on the permissions granted, users for the file and directory can be classified into one of these three groups:

- Owner
- Other users of the group
- All other users

2.4 Overview of Hadoop Application Protection

This section describes the various levels of protection provided by Hadoop Application Protection.

2.4.1 Protection in MapReduce Jobs

A MapReduce job in the Hadoop cluster involves sensitive data. You can use Protegrity interfaces to protect data when it is saved or retrieved from a protected source. The output data written by the job can be encrypted or tokenized. The protected data can be subsequently used by other jobs in the cluster in a secured manner. Field level data can be secured and ingested into HDFS by independent Hadoop jobs or other ETL tools.

For more information about secure ingestion of data in Hadoop, refer to section *2.6.2 Ingesting Files Using Hive Staging*.

For more information on the list of available APIs, refer to section *4.4 MapReduce APIs*.

If Hive queries are created to operate on sensitive data, then you can use Protegrity Hive UDFs for securing data. While inserting data to Hive tables, or retrieving data from protected Hive table columns, you can call Protegrity UDFs loaded into Hive during installation. The UDFs protect data based on the input parameters provided.

Secure ingestion of data into HDFS to operate Hive queries can be achieved by independent Hadoop jobs or other ETL tools.

For more information about securely ingesting data in Hadoop, refer to section *2.6 Ingesting Data Securely.*

2.4.2 Protection in Hive Queries

Protection in Hive queries is done by Protegrity Hive UDFs, which translates a HiveQL query into a MapReduce program and then sends it to the Hadoop cluster.

For more information on the list of available UDFs, refer to section *4.5 Hive UDFs*.

2.4.3 Protection in Pig Jobs

Protection in Pig jobs is done by Protegrity Pig UDFs, which are similar in function to the Protegrity UDFs in Hive.

For more information on the list of available UDFs, refer to section *4.6 Pig UDFs*.

2.4.4 Protection in HBase

HBase is a database which provides random read and write access to tables, consisting of rows and columns, in real-time. HBase is designed to run on commodity servers, to automatically scale as more servers are added, and is fault tolerant as data is divided across servers in the cluster. HBase tables are partitioned into multiple regions. Each region stores a range of rows in the table. Regions contain a datastore in memory and a persistent datastore(HFile). The Name node assigns multiple regions to a region server. The Name node manages the cluster and the region servers store portions of the HBase tables and perform the work on the data.

The Protegrity HBase protector extends the functionality of the data storage framework and provides transparent data protection and unprotection using coprocessors, which provide the functionality to run code directly on region servers. The Protegrity coprocessor for HBase runs on the region servers and protects the data stored in the servers. All clients which work with HBase are supported.

The data is transparently protected or unprotected, as required, utilizing the coprocessor framework.

For more information about HBase, refer to section *6 HBase*.

2.4.5 Protection in Impala

Impala is an MPP SQL query engine for querying the data stored in a cluster. It provides the flexibility of the SQL format and is capable of running the queries on HDFS in HBase.

The Protegrity Impala protector extends the functionality of the Impala query engine and provides UDFs which protect or unprotect the data as it is stored or retrieved.

For more information about the Impala protector, refer to section *7 Impala*.

2.4.6 Protection in HAWQ

HAWQ is an MPP database, which enable it to achieve data and processing parallelism.

The Protegrity HAWQ protector provides UDFs for protecting data using encryption or tokenization, and unprotecting data by using decryption or detokenization.

For more information about the HAWQ protector, refer to section *8 HAWQ*.

2.4.7 Protection in Spark

Spark is an execution engine that carries out batch processing of jobs in-memory and handles a wider range of computational workloads. In addition to processing a batch of stored data, Spark is capable of manipulating data in real time.

The Protegrity Spark protector extends the functionality of the Spark engine and provides APIs that protect or unprotect the data as it is stored or retrieved.

For more information about the Spark protector, refer to section *9 Spark*.

2.5 HDFS File Protection (HDFSFP)

Files are stored and retrieved by Hadoop system elements, such as file shell commands, MapReduce, Hive, Pig, HBase and so on. The stored files reside in HDFS and span multiple cluster nodes.

Most of the files in HDFS are plain text files and stored in the clear, with access control like a POSIX file system. These files contain sensitive data, making it vulnerable with exposure to unwanted users. These files are transparently protected as they are stored in HDFS. In addition, the content is exposed only to authorized users. The content in the files is unprotected transparently to processes or users, authorized to view and process the files. The user is automatically detected from the job information provided by HDFSFP. The job accessing secured files must be initialized by an authorized user having the required privileges in ACL. The files encrypted by HDFSFP are suitable for distributed processing by Hadoop distributed jobs like MapReduce.

HDFSFP protects individual files or files stored in a directory. The access control is governed by the security policy and ACL supplied by the security officer. The access control and security policy is controlled through ESA interfaces. Command line and UI options are available to control ACL entries for file paths and directories.

2.6 Ingesting Data Securely

This section describes the ways in which data can be secured and ingested by various jobs in Hadoop at a field or file level.

2.6.1 Ingesting Data Using ETL Tools and File Protector Gateway (FPG)

Protegrity provides the File Protector Gateway (FPG) for secure field level protection to ingest data through ETL tools. The ingested files data can be used by Hadoop applications for analytics and processing. The sensitive fields are secured by the FPG before Hadoop jobs operate on it.

For more information about field level ingestion by custom MapReduce job for data at rest in HDFS, refer to *File Protector Gateway Server Guide 6.6.4*.

2.6.2 Ingesting Files Using Hive Staging

Semi-structured data files can be loaded into a Hive staging table for ingestion into a Hive table with Hive queries and Protegrity UDFs. After loading data in the table, the data will be stored in protected form.

2.6.3 Ingesting Files into HDFS by HDFSFP

The HDFSFP component of Big Data Protector can be used for ingesting files securely in HDFS. It provides granular access control for the files in HDFS. You can ingest files using the command shell and Java API in HDFSFP.

For more information about using HDFSFP, refer to section *5 HDFS File Protector (HDFSFP)*.

2.7 Data Security Policy and Protection Methods

A data security policy establishes processes to ensure the security and confidentiality of sensitive information. In addition, the data security policy establishes administrative and technical safeguards against unauthorized access or use of the sensitive information.

Depending on the requirements, the data security policy typically performs the following functions:

- Classifies the data that is sensitive for the organization.

- Defines the methods to protect sensitive data, such as encryption and tokenization.
- Defines the methods to present the sensitive data, such as masking the display of sensitive information.
- Defines the access privileges of the users that would be able to access the data.
- Defines the time frame for privileged users to access the sensitive data.
- Enforces the security policies at the location where sensitive data is stored.
- Provides a means of auditing authorized and unauthorized accesses to the sensitive data. In addition, it can also provide a means of auditing operations to protect and unprotect the sensitive data.

The data security policy contains a number of components, such as, data elements, datastores, member sources, masks, and roles. The following list describes the functions of each of these entities:

- **Data elements** define the data protection properties for protecting sensitive data, consisting of the data securing method, data element type and its description. In addition, Data elements describe the tokenization or encryption properties, which can be associated with roles.
- **Datastores** consist of enterprise systems, which might contain the data that needs to be processed, where the policy is deployed and the data protection function is utilized.
- **Member sources** are the external sources from which users (or members) and groups of users are accessed. Examples are a file, database, LDAP, and Active Directory.
- **Masks** are a pattern of symbols and characters, that when imposed on a data field, obscures its actual value to the user. Masks effectively aid in hiding sensitive data.
- **Roles** define the levels of member access that are appropriate for various types of information. Combined with a data element, roles determine and define the unique data access privileges for each member.

For more information about the data security policies, protection methods, and the data elements supported by the components of the Big Data Protector, refer to *Protection Methods Reference Guide 6.6.5*.

3 Installing and Uninstalling Big Data Protector

This section describes the procedure to install and uninstall the Big Data Protector.

3.1 Installing Big Data Protector on a Cluster

This section describes the tasks for installing Big Data Protector on a cluster.

Starting from the Big Data Protector 6.6.4 release, you do not require *root* access to install Big Data Protector on a cluster.

You need a *sudoer* user account to install Big Data Protector on a cluster.

3.1.1 Verifying Prerequisites for Installing Big Data Protector

Ensure that the following prerequisites are met, before installing Big Data Protector:

- The Hadoop cluster is installed, configured, and running.
- ESA appliance version 6.6.5 is installed, configured, and running.
- A *sudoer* user account with privileges to perform the following tasks:
  - Update the system by modifying the configuration, permissions, or ownership of directories and files.
  - Perform third party configuration.
  - Create directories and files.
  - Modify the permissions and ownership for the created directories and files.
  - Set the required permissions to the create directories and files for the Protegrity Service Account.
  - Permissions for using the SSH service.
- The *sudoer* password is the same across the cluster.
- The following user accounts to perform the required tasks:
  - *ADMINISTRATOR_USER*: It is the *sudoer* user account that is responsible to install and uninstall the Big Data Protector on the cluster.
    This user account needs to have *sudo* access to install the product.
  - *EXECUTOR_USER*: It is a user that has ownership of all Protegrity files, folders, and services.
  - *OPERATOR_USER*: It is responsible for performing tasks such as, starting or stopping tasks, monitoring services, updating the configuration, and maintaining the cluster while the Big Data Protector is installed on it.
    If you need to start, stop, or restart the Protegrity services, then you need *sudoer* privileges for this user to impersonate the *EXECUTOR_USER*.

    Depending on the requriements, a single user on the system may perform multiple roles.

    If a single user is performing multiple roles, then ensure that the following conditions are met:

    - The user has the required permissions and privileges to impersonate the other user accounts, for performing their roles, and perform tasks as the impersonated user.
    - The user is assigned the highest set of privileges, from the required roles that it needs to perform, to execute the required tasks.
      For instance, if a single user is performing tasks as *ADMINISTRATOR_USER*, *EXECUTOR_USER,* and

OPERATOR_USER, then ensure that the user is assigned the privileges of the *ADMINISTRATOR_USER*.

- The management scripts provided by the installer in the *cluster_utils* directory should be run only by the user (*OPERATOR_USER*) having privileges to impersonate the *EXECUTOR_USER*.
  - If the value of the *AUTOCREATE_PROTEGRITY_IT_USR* parameter in the *BDP.config* file is set to *No*, then ensure that a service group containing a user for running the Protegrity services on all the nodes in the cluster already exists.
  - If the Hadoop cluster is configured with LDAP or AD for user management, then ensure that the *AUTOCREATE_PROTEGRITY_IT_USR* parameter in the *BDP.config* file is set to *No* and that the required service account user is created on all the nodes in the cluster.
- If the Big Data Protector with versions lower than 6.6.3 was previously installed with HDFSFP, then ensure that you create the backup of DFSFP on the ESA.
  For more information about creating the DFSFP backup, refer to section *4.1.4 Backing Up DFSFP before Installing Big Data Protector 6.6.3* in *Data Security Platform Upgrade Guide 6.6.5*.
- If Big Data Protector, version 6.6.3, with build version 6.6.3.15, or lower, was previously installed and the following Spark protector APIs for Encryption/Decryption are utilized:
  - *public void protect(String dataElement, List<Integer> errorIndex, short[] input, byte[][] output)*
  - *public void protect(String dataElement, List<Integer> errorIndex, int[] input, byte[][] output)*
  - *public void protect(String dataElement, List<Integer> errorIndex, long[] input, byte[][] output)*
  - *public void unprotect(String dataElement, List<Integer> errorIndex, byte[][] input, short[] output)*
  - *public void unprotect(String dataElement, List<Integer> errorIndex, byte[][] input, int[] output)*
  - *public void unprotect(String dataElement, List<Integer> errorIndex, byte[][] input, long[] output)*

  For more information, refer to the Advisory for Spark Protector APIs, before installing Big Data Protector, version 6.6.5.
- If the Big Data Protector was previously installed then uninstall it. In addition, delete the *<PROTEGRITY_DIR>* directory from the Lead node. If the */var/log/protegrity/* directory exists on any node in the cluster, then ensure that it is empty.
- Password based authentication is enabled in the *sshd_config* file before installation. After the installation is completed, this setting might be reverted back by the system administrator.
- The *lsb_release* library is present on the client machine, at least on the Lead node.
  The **Lead node** can be any node, such as the Name node, Data node, or Edge node, that can access the Hadoop cluster. The Lead node would be driving the installation of the Big Data Protector across the Hadoop cluster and is responsible for managing the Big Data Protector services throughout the cluster.
  If the *lsb_release* library is not present, then the installation of the Big Data Protector fails. This can be verified by running the following command.
  ```
  lsb_release
  ```
- If you are configuring the Big Data Protector with a Kerberos-enabled Hadoop cluster, then ensure that the HDFS superuser (*hdfs*) has a valid Kerberos ticket.
- If you are configuring HDFSFP with Big Data Protector, then ensure that the following prerequisites are met:
  - Ensure that an unstructured policy is created in the ESA, containing the data elements to be linked with the ACL.

- If a sticky bit is set for an HDFS directory, which is required to be protected by HDFSFP, then the user needs to remove the sticky bit before creating ACLs (for Protect/Reprotect/Unprotect/Update) for that HDFS directory. If required, then the user can set the sticky bit again after activating the ACLs.

For more information about creating data elements, security policies, and user roles, refer to *Enterprise Security Administrator Guide 6.6.5* and *Protection Enforcement Point Servers Installation Guide 6.6.5*.

3.1.2 Extracting Files from the Installation Package

➤ **To extract the files from the installation package:**

1. After receiving the installation package from Protegrity, copy it to the Lead node in any temporary folder, such as */opt/bigdata*.
2. Extract the files from the installation package using the following command:
   tar –xf BigDataProtector_<OS>-<arch>-nCPU_<Big data distribution>-64_6.6.5.x.tgz

   The following files are extracted:

   - *BDP.config*
   - *BdpInstallx.x.x_Linux_<arch>_6.6.5.x.sh*
   - *FileProtector_<OS>_x86-<arch>_AccessControl_6.6.x.x.sh*
   - *FileProtector_<OS>_x86-<arch>_ClusterDeploy_6.6.x.x.sh*
   - *FileProtector_<OS>_x86-<arch>_FileEncryption_6.6.x.x.sh*
   - *FileProtector_<OS>_x86-<arch>_PreInstallCheck_6.6.x.x.sh*
   - *FileProtector_<OS>_x86-<arch>_VolumeEncryption_6.6.x.x.sh*
   - *FP_ClusterDeploy_hosts*
   - *INSTALL.txt*
   - *JpepLiteSetup_Linux_<arch>_6.6.5.x.sh*
   - *node_uninstall.sh*
   - *PepHbaseProtectorx.x.xSetup_Linux_<arch>_<distribution>-x.x_6.6.5.x.sh*
   - *PepHdfsFp_Setup_<distribution>-x.x_6.6.5.x.sh*
   - *PepHivex.x.xSetup_Linux_<arch>_<distribution>-x.x_6.6.5.x.sh*
   - *PepImpalax.xSetup_<OS>_x86-<arch>_6.6.5.x.sh, only if it is a Cloudera or MapR distribution*
   - *PepHawqx.xSetup_<OS>_x86-<arch>_6.6.5.x.sh, only if it is a Pivotal distribution*
   - *PepMapreducex.x.xSetup_Linux_<arch>_<distribution>-x.x_6.6.5.x.sh*
   - *PepPigx.x.xSetup_Linux_<arch>_<distribution>-x.x_6.6.5.x.sh*
   - *PepServer_Setup_Linux_<arch>_6.6.5.x.sh*
   - *PepSparkx.x.xSetup_Linux_<arch>_<distribution>-x.x_6.6.5.x.sh*
   - *PepTalendSetup_x.x.x_6.6.5.x.sh*
   - *Prepackaged_Policyx.x.x_Linux_<arch>_6.6.5.x.sh*
   - *ptyLogAnalyzer.sh*
   - *ptyLog_Consolidator.sh*
   - *samples-mapreduce.tar*
   - *samples-spark.tar*
   - *uninstall.sh*
   - *XCPep2Jni_Setup_Linux_<arch>_6.6.5.x.sh*

3.1.3 Updating the BDP.config File

Ensure that the *BDP.config* file is updated before the Big Data Protector is installed.

Do not update the *BDP.config* file when the installation of the Big Data Protector is in progress.

➤ **To update the *BDP.config* file:**

1. Create a file containing a list of all nodes in the cluster, except the Lead node, and specify it in the *BDP.config* file.
   This file is used by the installer for installing Big Data Protector on the nodes.
2. Open the *BDP.config* file in any text editor and modify the following parameter values:
   - HADOOP_DIR – The installation home directory for the Hadoop distribution.
   - PROTEGRITY_DIR – The directory where the Big Data Protector will be installed.
     The samples and examples used in this document assume that the Big Data Protector is installed in the */opt/protegrity/* directory.
   - CLUSTERLIST_FILE – This file contains the host name or IP addresses all the nodes in the cluster, except the Lead node, listing one host name and IP address per line.
     Ensure that you specify the file name with the complete path.
   - INSTALL_DEMO – Specifies one of the following values, as required:
     - Yes – The installer installs the demo.
     - No – The installer does not install the demo.
   - HDFSFP – Specifies one of the following values, as required:
     - Yes – The installer installs HDFSFP.
     - No – The installer does not install HDFSFP.

     If HDFSFP is being installed, then XCPep2Jni is installed using the *XCPep2Jni_Setup_Linux_<arch>_6.6.5.x.sh* script.
   - SPARK_PROTECTOR – Specifies one of the following values, as required:
     Yes – The installer installs the Spark protector. This parameter also needs to be set to *Yes*, if the user needs to run Hive UDFs with Spark SQL, or use the Spark protector samples if the *INSTALL_DEMO* parameter is set to *Yes*.
     - No – The installer does not install the Spark protector.
   - IP_NN – The IP address of the Lead node in the Hadoop cluster, which is required for the installation of HDFSFP.
   - PROTEGRITY_CACHE_PORT – The Protegrity Cache port used in the cluster. This port should be open in the firewall across the cluster. On the Lead node, it should be open only for the corresponding ESA, which is used to manage the cluster protection. This is required for the installation of HDFSFP. Typical value for this port is 6379.
   - AUTOCREATE_PROTEGRITY_IT_USR – This parameter determines the Protegrity service account. The service group and service user name specified in the *PROTEGRITY_IT_USR_GROUP* and *PROTEGRITY_IT_USR* parameters respectively will be created if this parameter is set to *Yes*. One of the following values can be specified, as required:
     - Yes – The installer creates a service group *PROTEGRITY_IT_USR_GROUP* containing the user *PROTEGRITY_IT_USR* for running the Protegrity services on all the nodes in the cluster.
       If the service group or service user are already present, then the installer exits.
       If you uninstall the Big Data Protector, then the service group and the service user are deleted.

  - No – The installer does not create a service group *PROTEGRITY_IT_USR_GROUP* with the service user *PROTEGRITY_IT_USR* for running the Protegrity services on all the nodes in the cluster.

    Ensure that a service group containing a service user for running Protegrity services has been created, as described in section *3.1.1 Verifying Prerequisites for Installing Big Data Protector*.

- PROTEGRITY_IT_USR_GROUP – This service group is required for running the Protegrity services on all the nodes in the cluster. All the Protegrity installation directories are owned by this service group.
- PROTEGRITY_IT_USR – This service account user is required for running the Protegrity services on all the nodes in the cluster and is a part of the group *PROTEGRITY_IT_USR_GROUP*. All the Protegrity installation directories are owned by this service user.
- HADOOP_NATIVE_DIR – The Hadoop native directory. This parameter needs to be specified if you are using MapR.
- HADOOP_SUPER_USER – The Hadoop super user name. This parameter needs to be specified if you are using MapR.

3.1.4 Installing Big Data Protector

➤ **To install the Big Data Protector:**

1. As a *sudoer* user, run *BdpInstallx.x.x_Linux_<arch>_6.6.5.x.sh* from the folder where it is extracted.

   A prompt to confirm or cancel the Big Data Protector installation appears.
2. Type *yes* to continue with the installation.

   The Big Data Protector installation starts.

   If you are using a Cloudera or MapR distribution, then the presence of the HDFS connection is also verified.

   A prompt to enter the *sudoer* password for the *ADMINISTRATOR* user appears.
3. Enter the *sudoer* password.

   A prompt to enter the ESA user name or IP address appears.
4. Enter the ESA host name or IP address.

   A prompt to enter the ESA user name appears.
5. Enter the ESA user name (Security Officer).

   The PEP Server Installation wizard starts and a prompt to configure the host as ESA proxy appears.
6. Depending on the requirements, type *Yes* or *No* to configure the host as an ESA proxy.
7. If the ESA proxy is set to *Yes*, then enter the host password for the required ESA user.
8. When prompted, perform the following steps to download the ESA keys and certificates.
   a) Specify the Security Officer user with administrative privileges.
   b) Specify the Security Officer password for the ESA certificates and keys.

   The installer then installs the Big Data Protector on all the nodes in the cluster.

   The status of the installation of the individual components appears, and the log files for all the required components on all the nodes in the cluster are stored on the Lead node in the *<PROTEGRITY_DIR>/cluster_utils/logs* directory.

   Verify the installation report, that is generated at *<PROTEGRITY_DIR>/cluster_utils/installation_report.txt* to ensure that the installation of all the components is successful on all the nodes in the cluster.

   Verify the *bdp_setup.log* file confirm if the Big Data Protector was installed successfully on all the nodes in the cluster.

9. Restart the MapReduce (MRv1) or Yarn (MRv2) services on the Hadoop cluster.

   The installer installs the following components in the installation folder of the Big Data Protector:

   - PEP server in the *<PROTEGRITY_DIR>/defiance_dps* directory
   - XCPep2Jni in the *<PROTEGRITY_DIR>/defiance_xc* directory
   - JpepLite in the *<PROTEGRITY_DIR>/jpeplite* directory
   - MapReduce protector in the *<PROTEGRITY_DIR>/pepmapreduce/lib* directory
   - Hive protector in the *<PROTEGRITY_DIR>/pephive/lib* directory
   - Pig protector in the *<PROTEGRITY_DIR>/peppig/lib* directory
   - HBase protector in the *<PROTEGRITY_DIR>/pephbase-protector/lib* directory
   - Impala protector in the *<PROTEGRITY_DIR>/pepimpala* directory, if you are using a Cloudera or MapR distribution
   - HAWQ protector in the *<PROTEGRITY_DIR>/pephawq* directory, if you are using a Pivotal distribution
   - *hdfsfp-xxx.jar* in the *<PROTEGRITY_DIR>/hdfsfp* directory, only if the value of the *HDFSFP* parameter in the *BDP.config* file is specified as *Yes*
   - *pepspark-xxx.jar* in the *<PROTEGRITY_DIR>/pepspark/lib* directory, only if the value of the *SPARK* parameter in the *BDP.config* file is specified as *Yes*
   - Talend-related files in *<PROTEGRITY_DIR>/etl/talend* directory
   - Cluster Utilities in the *<PROTEGRITY_DIR>/cluster_utils* directory
     The following files and directories are present in the *<PROTEGRITY_DIR>/cluster_utils* folder:
     - *BdpInstallx.x.x_Linux_<arch>_6.6.5.x.sh* utility to install the Big Data Protector on any node in the cluster.

       For more information about using the *BdpInstallx.x.x_Linux_<arch>_6.6.5.x.sh* utility, refer to section *3.2.1 Installing Big Data Protector on New Nodes added to a Hadoop Cluster*.
     - *cluster_cachesrvctl.sh* utility for monitoring the status of the Protegrity Cache on all the nodes in the cluster, only if the value of the *HDFSFP* parameter in the *BDP.config* file is specified as *Yes*.
     - *cluster_pepsrvctl.sh* utility for managing PEP servers on all nodes in the cluster.
     - *uninstall.sh* utility to uninstall the Big Data Protector from all the nodes in the cluster.
     - *node_uninstall.sh* to uninstall the Big Data Protector from any nodes in the cluster.

       For more information about using the *node_uninstall.sh* utility, refer to section *3.2.2 Uninstalling Big Data Protector from Selective Nodes in the Hadoop Cluster*.
     - *update_cluster_policy.sh* utility for updating PEP servers when a new policy is deployed.
     - *BDP.config* file
     - *CLUSTERLIST_FILE*, which is a file containing a list of all the nodes, except the Lead node.
     - *installation_report.txt* file that contains the status of installation of all the components in the cluster.
     - *logs* directory that contains the consolidated setup logs from all the nodes in the cluster.

10. Starting with the Big Data Protector, version 6.6.4, the Bulk APIs in the MapReduce protector will return the detailed error and return codes instead of *0* for *failure* and *1* for *success*.

    For more information about the error codes for Big Data Protector, version 6.6.5, refer to *Table 11-2 PEP Log Return Codes* and *Table 11-3 PEP Result Codes* in section *11 Appendix: Return Codes*.

    If the older behaviour from the Big Data Protector, version 6.6.3 or lower with the Bulk APIs in the MapReduce protector is desired, then perform the following steps to enable the Backward compatibility mode to retain the same error handling capabilities.

    a) If you are using HDP, version 2.2 or higher (Hortonworks), or PHD, version 3.0 or higher (Pivotal Hadoop), then append the following entry to the *mapreduce.admin.reduce.child.java.opts* property in the *mapred-site.xml* file.

    -Dpty.mr.compatibility=old

    b) If you are using CDH, then add the following values to the *Yarn Service Mapreduce Advanced Configuration Snippet (Safety Valve)* parameter in the *mapred-site.xml* file.

    <property>
    <name>mapreduce.admin.map.child.java.opts</name>
    <value>-Dpty.mr.compatibility=old</value>
    </property>

    <property>
    <name>mapreduce.admin.reduce.child.java.opts</name>
    <value>-Dpty.mr.compatibility=old</value>
    </property>

11. If you are using HDP, version 2.2 or higher (Hortonworks), or PHD, version 3.0 or higher (Pivotal Hadoop), and you have installed HDFSFP, then perform the following steps.

    a) Ensure that the *mapreduce.application.classpath* property in the *mapred-site.xml* file contains the following entries in the order provided.

    *<PROTEGRITY_DIR>/pepmapreduce/lib/**

    *<PROTEGRITY_DIR>/pephive/lib/**

    *<PROTEGRITY_DIR>/peppig/lib/**

    *<PROTEGRITY_DIR>/hdfsfp/**

    Ensure that the above entries are before all other entries in the *mapreduce.application.classpath* property.

    b) Ensure that the *mapred.min.split.size* property in the *hive-site.xml* file is set to the following value.

    mapred.min.split.size=256000

    c) Restart the Yarn service.

    d) Restart the MRv2 service.

    e) Ensure that the *tez.cluster.additional.classpath.prefix* property in the *tez-site.xml* file contains the following entries in the order provided.

    *<PROTEGRITY_DIR>/pepmapreduce/lib/**

    *<PROTEGRITY_DIR>/pephive/lib/**

*<PROTEGRITY_DIR>/peppig/lib/**

*<PROTEGRITY_DIR>/hdfsfp/**

Ensure that the above entries are before all other entries in the *tez.cluster.additional.classpath.prefix* property.

f) Restart the Tez services.

12. If you are using HDP, version 2.2 or higher (Hortonworks), or PHD, version 3.0 or higher (Pivotal Hadoop), and you have not installed HDFSFP, then perform the following steps.
    a) Ensure that the *mapreduce.application.classpath* property in the *mapred-site.xml* file contains the following entries.

    *<PROTEGRITY_DIR>/pepmapreduce/lib/**

    *<PROTEGRITY_DIR>/pephive/lib/**

    *<PROTEGRITY_DIR>/peppig/lib/**

    Ensure that the above entry is before all other entries in the *mapreduce.application.classpath* property.

    b) Ensure that the *yarn.application.classpath* property in the *yarn-site.xml* file contains the following entries.

    *<PROTEGRITY_DIR>/pepmapreduce/lib/**

    *<PROTEGRITY_DIR>/pephive/lib/**

    *<PROTEGRITY_DIR>/peppig/lib/**

    Ensure that the above entry is before all other entries in the *yarn.application.classpath* property.

    c) Restart the Yarn service.

    d) Restart the MRv2 service.

    e) Ensure that the *tez.cluster.additional.classpath.prefix* property in the *tez-site.xml* file contains the following entries.

    *<PROTEGRITY_DIR>/pepmapreduce/lib/**

    *<PROTEGRITY_DIR>/pephive/lib/**

    *<PROTEGRITY_DIR>/peppig/lib/**

    Ensure that the above entry is before all other entries in the *tez.cluster.additional.classpath.prefix* property.

    f) Restart the Tez services.

13. If HDFSFP is not installed and you need to use the Hive protector, then perform the following steps.
    a) Specify the following value for the *hive.exec.pre.hooks* property in the *hive-site.xml* file.

    ```
    hive.exec.pre.hooks=com.protegrity.hive.PtyHiveUserPreHook
    ```

    b) Restart the Hive services to ensure that the updates are propagated to all the nodes in the cluster.

14. If HDFSFP is installed and you need to use the Hive protector with HDFSFP, then perform the following steps.
    a) Specify the following value for the *hive.exec.pre.hooks* property in the *hive-site.xml* file.

    ```
    hive.exec.pre.hooks=com.protegrity.hadoop.fileprotector.hive.PtyHivePreHook
    ```

    b) Restart the Hive services to ensure that the updates are propagated to all the nodes in the cluster.

If you are using Beeline or Hue, then ensure that Protegrity Big Data Protector is installed on the following machines:

- For Beeline: The machines where Hive Metastore, and HiveServer2 are running.
- For Hue: The machines where HueServer, Hive Metastore, HiveServer2 are running.

It is recommended to use the Cluster Policy provider to deploy the policies in a multi-node cluster environment, such as Big Data, Teradata etc.

If you require the PEP Server service to start automatically after every reboot of the system, then define the PEP Server service in the startup with the required run levels.

For more info about starting the PEP Server service automatically, refer to *Protection Enforcements Point Servers Installation Guide Release 6.6.5*.

3.1.5 Applying Patches

As the functionality of the ESA is extended, it should be updated through patches applied to ESA. The patches are available as *.pty* files, which should be loaded with the ESA user interface.

Receive the *ESA_PAP-ALL-64_x86-64_6.6.5.pty*, or later patch from Protegrity. Upload this patch on the ESA using the Web UI. Then install this patch using the ESA CLI manager.

For more information about applying patches, refer to section *4.4.6.2 Install Patches* of *Protegrity Appliances Overview*.

3.1.6 Installing the DFSFP Service

Using the **Add/Remove Services** tool on the ESA to install the DFSFP service.

For more information about installing services, refer to Section 4.4.6 of *Protegrity Appliances Overview*.

➤ **To install the DFSFP service using the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Administration➤ Add/Remove Services**.
3. Press ENTER.
   The *root* password prompt appears.
4. Enter the *root* password.
5. Press ENTER.
   The Add/Remove Services screen appears
6. Select **Install applications**.
7. Press ENTER.
8. Select DFSFP.
9. Press ENTER.
   The DFSFP service is installed.

3.1.7 Configuring HDFSFP

If HDFSFP is used, then it should be configured after Big Data Protector is installed. To ensure that the user is able to access protected data in the Hadoop cluster, HDFSFP is globally configured so that it can perform checks for access control transparently.

Ensure that you set the the value of the *mapreduce.output.fileoutputformat.compress.type* property to *BLOCK* in the *mapred-site.xml* file.

3.1.7.1 Configuring HDFSFP for Yarn (MRv2)

➤ **To configure Yarn (MRv2) with HDFSFP:**

1. Register the Protegrity codec in the Hadoop codec factory configuration. In the *io.compression.codecs* property in the *core-site.xml* file, add the codec **com.protegrity.hadoop.fileprotector.crypto.codec.PtyCryptoCodec**.
2. Modify the value of the *mapreduce.output.fileoutputformat.compress* property in the *mapred-site.xml* file to *true*.
3. Add the property *mapreduce.output.fileoutputformat.compress.codec* to the *mapred-site.xml* file and set the value to **com.protegrity.hadoop.fileprotector.crypto.codec.PtyCryptoCodec**.
   If the property is already present in the *mapred-site.xml* file, then ensure that the existing value of the property is replaced with **com.protegrity.hadoop.fileprotector.crypto.codec.PtyCryptoCodec**.
4. Include the *<PROTEGRITY_DIR>/hdfsfp/** path as the first value in the *yarn.application.classpath* property in the *yarn-site.xml* file.
5. Restart the HDFS and Yarn services.

3.1.7.2 Configuring HDFSFP for MapReduce, v1 (MRv1)

A MapReduce job processes large data sets stored in HDFS across the Hadoop cluster. The result of the MapReduce job is stored in HDFS. The HDFSFP stores protected data in encrypted form in HDFS.

The Map job reads protected data and the Reduce job saves the result in protected form. This is done by configuring the Protegrity codec at global level for MapReduce jobs.

➤ **To configure MRv1 with HDFSFP:**

1. Register the Protegrity codec in the Hadoop codec factory configuration. In the *io.compression.codecs* property in the *core-site.xml* file, add the codec **com.protegrity.hadoop.fileprotector.crypto.codec.PtyCryptoCodec**.
2. Modify the value of the *mapred.output.compress* property in the *mapred-site.xml* file to *true*.
3. Modify the value of the *mapred.output.compression.codec* property in the *mapred-site.xml* file to **com.protegrity.hadoop.fileprotector.crypto.codec.PtyCryptoCodec**.
4. Restart the HDFS and MapReduce services.

3.1.7.3 Adding a Cluster to the ESA

Before configuring the Cache Refresh Server, ensure that a cluster is added to the ESA.

For more information about adding a cluster to the ESA, refer to section *5.14.1 Adding a Cluster for Protection*.

3.1.7.4 Configuring the Cache Refresh Server

If a cluster is added to the ESA, then the Cache Refresh server periodically validates the cache entries and takes corrective action, if necessary. This server should always be active.

The Cache Refresh Server periodically validates the ACL entries in Protegrity Cache with the ACL entries in the ESA.

- If a Data store is created using ESA 6.5 SP2 Patch 3 with DFSFPv3 patch installed, then the Cluster configuration file (*clusterconfig.xml*), located in the *<PROTEGRITY_DIR>/dfs/dfsadmin/config/* directory, contains the field names *RedisPort* and *RedisAuth*.
- If a Data store is created using ESA 6.5 SP2 Patch 4 with DFSFPv8 patch installed, then the Cluster configuration file (*clusterconfig.xml*) contains the field names *ProtegrityCachePort* and *ProtegrityCacheAuth*.
- If a migration of the ESA 6.5 SP2 Patch 3 with DFSFPv3 patch installed to the ESA 6.5 SP2 Patch 4 with DFSFPv8 patch installed is done, then the Cluster configuration file (*clusterconfig.xml*) contains the field name entries *RedisPort* and *RedisAuth* for the old Data stores, and the entries *ProtegrityCachePort* and *ProtegrityCacheAuth* for the new Data stores, created after the migration.

If the ACL entries present in the appliance are not matching the ACL entries in Protegrity Cache, then logs are generated in the ESA. The logs can be viewed from the ESA Web Interface at the following path: **Distributed File System File Protector➤ Logs**.

The various error codes are explained in *Troubleshooting Guide 6.6.5*.

➤ **To configure the Cache Refresh Server time:**

1. Navigate to the path *<PROTEGRITY_DIR>/dfs/cacherefresh/data*.
2. Open the *dfscacherefresh.cfg* file.
3. Modify the *cacherefreshtime* parameter as required based on the following guidelines:
   - Default value – 30 minutes
   - Minimum value – 10 minutes
   - Maximum value – 720 minutes (12 hours)

The Cache Refresh Interval should be entered in minutes.

➤ **To verify if the Cache Refresh Server is running:**

1. Login to the ESA Web Interface.
2. Navigate to **System➤ Services➤ DFS Cache Refresh**.
   The Cache Refresh Server would be running.
3. If the Cache Refresh Server is not running, then click on the **Start** button ( ) to start the Cache Refresh Server.

3.1.7.5 Configuring Hive Support in HDFSFP

If Hive is used with HDFSFP, then it should be configured after installing Big Data Protector.

➤ **To configure Hive support in HDFSFP:**

1. If you are using a Hadoop distribution that has a Management UI, then perform the following steps.
   a) In the *hive-site.xml* file, set the value of the *mapreduce.job.maps* property to *1*, using the Management UI.

If the *hive-site.xml* file does not have any *mapreduce.job.maps* property, then perform the following tasks.

a. Add the property with the name *mapreduce.job.maps* in the *hive-site.xml* file.

b. Set the value of the *mapreduce.job.maps* property to *1*.

b) In the *hive-site.xml* file, add the value *com.protegrity.hadoop.fileprotector.hive.PtyHivePreHook* to the *hive.exec.pre.hooks* property before any other existing value, using the Management UI.

If the *hive-site.xml* file does not have any *hive.exec.pre.hooks* property, then perform the following tasks.

a. Add the property with the name *hive.exec.pre.hooks* in the *hive-site.xml* file.

b. Set the value of the *hive.exec.pre.hooks* property to *com.protegrity.hadoop.fileprotector.hive.PtyHivePreHook*.

2. If you are using a Hadoop distribution without a Management UI, then perform the following steps.

a) Add the following property in the *hive-site.xml* file on all nodes.

```
<property>
<name>mapreduce.job.maps</name>
<value>1</value>
</property>
```

If the property is already present in the *hive-site.xml* file, then ensure that the value of the property is set to *1*.

b) Add the following property in the *hive-site.xml* file on all nodes.

```
<property>
<name>hive.exec.pre.hooks</name>
<value>com.protegrity.hadoop.fileprotector.hive.PtyHivePreHook</value>
</property>
```

If the property is already present in the *hive-site.xml* file, then ensure that the value *com.protegrity.hadoop.fileprotector.hive.PtyHivePreHook* is before any other existing value.

For more information about using Hive with HDFSFP, refer to section *13 Appendix: Using Hive with HDFSFP.*

3.1.8 Configuring HBase

If HBase is used, then it should be configured after Big Data Protector is installed.

Ensure that you configure the Protegrity HBase coprocessor on all the region servers. If the Protegrity HBase coprocessor is not configured in some region servers, then an inconsistent state might occur, where some records in a table are protected and some are not protected.

This could potentially lead to data corruption, making it difficult to separate the protected data from clear text data.

It is recommended to use HBase version 0.98 or above.

If you are using an HBase version lower than 0.98, then you would need a Java client to perform the protection of data. HBase versions lower than 0.98 do not support ATTRIBUTES, which controls the MIGRATION and BYPASS_COPROCESSOR parameters.

➤ **To configure HBase:**

1. If you are using a Hadoop distribution that has a Management UI, then add the following value to the HBase coprocessor region classes property in the *hbase-site.xml* file in all the respective region server groups, using the Management UI.
   ```
   com.protegrity.hbase.PTYRegionObserver
   ```
   If the *hbase-site.xml* file does not have any HBase coprocessor region classes property, then perform the following tasks.
   a) Add the property with the name *hbase.coprocessor.region.classes* in the *hbase-site.xml* file in all the respective region server groups.
   b) Set the following value for the *hbase.coprocessor.region.classes* property.
   ```
   com.protegrity.hbase.PTYRegionObserver
   ```
   
   If any coprocessors are already defined in the *HBase coprocessor region class* property, then ensure that the value of the Protegrity coprocessor is before any pre-existing coprocessors defined in the *hbase-site.xml* file.
2. If you are using a Hadoop distribution without a Management UI, then add the following property in the *hbase-site.xml* file on all region server nodes.
   ```
   <property>
   <name>hbase.coprocessor.region.classes</name>
   <value>com.protegrity.hbase.PTYRegionObserver</value>
   </property>
   ```
   If the property is already present in the *hbase-site.xml* file, then ensure that the value of the Protegrity coprocessor region class is before any other coprocessor in the *hbase-site.xml* file.
3. Restart all HBase services.

3.1.9 Configuring Impala

If Impala is used, then it should be configured after Big Data Protector is installed.

➤ **To configure Impala:**

1. Ensure that the Hadoop cluster is installed, configured, and running.
2. Navigate to the *<PROTEGRITY_DIR>/pepimpala/sqlscripts/* folder.
   This folder contains the Protegrity UDFs for the Impala protector.
3. If you are not using a Kerberos-enabled Hadoop cluster, then execute the *createobjects.sql* script to load the Protegrity UDFs for the Impala protector.
   impala-shell -i <IP address of any Impala slave node> -f <PROTEGRITY_DIR>/pepimpala/sqlscripts/createobjects.sql
4. If you are using a Kerberos-enabled Hadoop cluster, then execute the *createobjects.sql* script to load the Protegrity UDFs for the Impala protector.
   impala-shell -i <IP address of any Impala slave node> -f <PROTEGRITY_DIR>/pepimpala/sqlscripts/createobjects.sql -k

   
   If the *catalogd* process is restarted at any point in time, then all the Protegrity UDFs for the Impala protector should be reloaded using the command in *Step 3 or 4, as required*.

3.1.10 Configuring HAWQ

If HAWQ is used, then it should be configured after Big Data Protector is installed.

Ensure that you are logged as the *gpadmin* user for configuring HAWQ.

To configure HAWQ:

1. Ensure that the Hadoop cluster is installed, configured, and running.
2. Navigate to the *<PROTEGRITY_DIR>/pephawq/sqlscripts/* folder.
   This folder contains the Protegrity UDFs for the HAWQ protector.
3. Execute the *createobjects.sql* script to load the Protegrity UDFs for the HAWQ protector.
   psql -h <HAWQ_Master_Hostname> -p 5432 -f <PROTEGRITY_DIR>/pephawq/sqlscripts/createobjects.sql
   where:
   HAWQ_Master_Hostname: Hostname or IP Address of the HAWQ Master Node
   5432: Port number

3.1.11 Configuring Spark

If Spark is used, then it should be configured after Big Data Protector is installed.

To configure Spark:

1. Ensure that the Hadoop cluster is installed, configured, and running.
2. Update the *spark-defaults.conf* file to include the following classpath entries, using Hadoop services, Cloudera Manager for Cloudera distributions, or Ambari Server for Hortonworks or Pivotal distributions, depending on the environment.
   spark.driver.extraClassPath=<PROTEGRITY_DIR>/pepspark/lib/*
   spark.executor.extraClassPath=<PROTEGRITY_DIR>/pepspark/lib/*
3. If HDFSFP is installed, then update the *spark-defaults.conf* file to include the following classpath entries.
   spark.driver.extraClassPath=<PROTEGRITY_DIR>/pepspark/lib/*:<PROTEGRITY_DIR>/hdfsfp/*
   spark.executor.extraClassPath=<PROTEGRITY_DIR>/pepspark/lib/*:<PROTEGRITY_DIR>/hdfsfp/*
4. Save the *spark-defaults.conf* file.
5. Deploy the configuration change to all the nodes in the Hadoop cluster.
6. Restart the Spark services.

If the user needs to run Hive UDFs with Spark SQL, then the following steps need to be performed.

To configure Spark SQL:

1. Ensure that the Hadoop cluster is installed, configured, and running.
2. Update the *spark-defaults.conf* file to include the following classpath entries, using Hadoop services, Cloudera Manager for Cloudera distributions, or Ambari Server for Hortonworks or Pivotal distributions, depending on the environment.
   spark.driver.extraClassPath=<PROTEGRITY_DIR>/pephive/lib/*:<PROTEGRITY_DIR>/pepspark/lib/*
   spark.executor.extraClassPath=<PROTEGRITY_DIR>/pephive/lib/*:<PROTEGRITY_DIR>/pepspark/lib/*
3. If HDFSFP is installed, then update the *spark-defaults.conf* file to include the following classpath entries.

spark.driver.extraClassPath=<PROTEGRITY_DIR>/pephive/lib/*:<PROTEGRITY_DIR>/pepspark/lib/*:<PROTEGRITY_DIR>/hdfsfp/*

spark.executor.extraClassPath=<PROTEGRITY_DIR>/pephive/lib/*:<PROTEGRITY_DIR>/pepspark/lib/*:<PROTEGRITY_DIR>/hdfsfp/*

4. Save the *spark-defaults.conf* file.
5. Deploy the configuration change to all the nodes in the Hadoop cluster.
6. Restart the Spark services.

3.2 Installing or Uninstalling Big Data Protector on Specific Nodes

This section describes the following procedures:

- Installing Big Data Protector on New Nodes added to a Hadoop cluster
- Uninstalling Big Data Protector from a Nodes in the Hadoop cluster

3.2.1 Installing Big Data Protector on New Nodes added to a Hadoop Cluster

If you need to install Big Data Protector on new nodes added to a Hadoop cluster, then use the *BdpInstallx.x.x_Linux_<arch>_6.6.5.x.sh* utility in the *<PROTEGRITY_DIR>/cluster_utils* directory.

Ensure that you install the Big Data Protector from an *ADMINISTRATOR* user having full *sudoer* privileges.

➤ **To install Big Data Protector on New Nodes added to a Hadoop Cluster:**

1. Login to the Lead Node.
2. Navigate to the *<PROTEGRITY_DIR>/cluster_utils* directory.
3. Add additional entries for each new node, on which the Big Data Protector needs to be installed, in the *NEW_HOSTS_FILE* file.

   The new nodes from the *NEW_HOSTS_FILE* file will be appended to the *CLUSTERLIST_FILE*.
4. Execute the following command utility to install Big Data Protector on the new nodes.

   ./BdpInstall1.0.1_Linux_<arch>_6.6.5.X.sh –a <NEW_HOSTS_FILE>

   The Protegrity Big Data Protector is installed on the new nodes.

3.2.2 Uninstalling Big Data Protector from Selective Nodes in the Hadoop Cluster

If you need to uninstall Big Data Protector from selective nodes in the Hadoop cluster, then use the *node_uninstall.sh* utility in the *<PROTEGRITY_DIR>/cluster_utils* directory.

Ensure that you uninstall the Big Data Protector from an *ADMINISTRATOR* user having full *sudoer* privileges.

➤ **To uninstall Big Data Protector from Selective Nodes in the Hadoop Cluster:**

1. Login to the Lead Node.
2. Navigate to the *<PROTEGRITY_DIR>/cluster_utils* directory.
3. Create a new hosts file (such as *NEW_HOSTS_FILE*).

   The *NEW_HOSTS_FILE* file contains the required nodes on which the Big Data Protector needs to be uninstalled.
4. Add the nodes from which the Big Data Protector needs to be uninstalled in the new hosts file.

5. Execute the following command to remove the Big Data Protector from the nodes that are listed in the new hosts file.
   ./node_uninstall.sh -c NEW_HOSTS_FILE
   The Big Data Protector is uninstalled from the nodes listed in the new hosts file.
6. Remove the nodes from which the Big Data Protector is uninstalled in Step 5 from the CLUSTERLIST_FILE file.

3.3 Utilities

This section provides information about the following utilities:

- PEP Server Control (*cluster_pepsrvctl.sh*) – Manages PEP servers across the cluster.
- Update Cluster Policy (*update_cluster_policy.sh*) – Updates the configurations of the PEP servers across the cluster.
- Protegrity Cache Control (*cluster_cachesrvctl.sh*) – Monitors the status of the Protegrity Cache on all the nodes in the cluster. This utility is available only for HDFSFP.
- Recover Utility – Recovers the contents from a protected path. This utility is available only for HDFSFP.

Ensure that you run the utilities with a user (*OPERATOR_USER*) having sudo privileges for impersonating the service account (*EXECUTOR_USER* or *PROTEGRITY_IT_USR*, as configured).

3.3.1 PEP Server Control

This utility (*cluster_pepsrvctl.sh*), in the *<PROTEGRITY_DIR>/cluster_utils* folder, manages the PEP server services on all the nodes in the cluster, except the Lead node.

The utility provides the following options:

- Start – Starts the PEP servers in the cluster.
- Stop – Stops the PEP servers in the cluster.
- Restart – Restarts the PEP servers in the cluster.
- Status – Reports the status of the PEP servers.

The utility (*pepsrvctrl.sh*), in the *<PROTEGRITY_DIR>/defiance_dps/bin/* folder, manages the PEP server services on the Lead node.

When you run the the PEP Server Control utility, then you will be prompted to enter the *OPERATOR_USER* password, which is same across all the nodes in the cluster.

3.3.2 Update Cluster Policy

This utility (*update_cluster_policy.sh*), in the *<PROTEGRITY_DIR>/cluster_utils* folder, updates the configurations of the PEP servers across the cluster.

For example, if you need to make any changes to the PEP server configuration, make the changes on the Lead node and then propagate the change to all the PEP servers in the cluster using the *update_cluster_policy.sh* utility.

Ensure that all the PEP servers in the cluster are stopped before running the *update_cluster_policy.sh* utility.

When you run the the Update Cluster Policy utility, then you will be prompted to enter the *OPERATOR_USER* password, which is same across all the nodes in the cluster.

3.3.3 Protegrity Cache Control

This utility (*cluster_cachesrvctl.sh*), in the *<PROTEGRITY_DIR>/cluster_utils* folder, monitors the status of the Protegrity Cache on all the nodes in the cluster. This utility prompts for the *OPERATOR_USER* password.

The utility provides the following options:

- Start – Starts the Protegrity Cache services in the cluster.
- Stop – Stops the Protegrity Cache services in the cluster.
- Restart – Restarts the Protegrity Cache services in the cluster.
- Status – Reports the status of the Protegrity Cache services.

3.3.4 Recover Utility

The Recover utility is available for HDFSFP only. This utility recovers the contents from protected files of types Text, RC, and Sequence, in the absence of ACL or loss of ACL information. This ensures that the data is not lost under any circumstances.

```
<PROTEGRITY_DIR>/hdfsfp/recover.sh -srcpath [HDFS path] -destpath [Local File System Directory]
```

Parameters

`srcpath:` The protected HDFS path containg the data to be unprotected.
`destpath:` The destination directory to store unprotected data.

Result

- If *srcpath* is the file path, then the Recover utility recovers all files.
- If *srcpath* is the directory path, then the Recover utility recovers all files inside the directory.

Ensure that the user running the Recover utility has unprotect access on the data element which was used to protect the files in the HDFS path.

Ensure that an *ADMINISTRATOR* or *OPERATOR_USER* is running the Recover Utility and the user has the required read/execute permissions to the *<PROTEGRITY_DIR>/hdfsfp/recover.sh* script.

Example

The following two ACLs are created:

1. */user/root/employee*
2. */user/ptyitusr/prot/employee*

Run the Recover Utility on these two paths with destination local directory as */tmp/HDFSFP-recovered/* by using the following commands.

```
<PROTEGRITY_DIR>/hdfsfp/recover.sh -srcpath /user/root/employee -destpath /tmp/HDFSFP-recovered/

<PROTEGRITY_DIR>/hdfsfp/recover.sh -srcpath /user/ptyitusr/prot/employee -destpath /tmp/HDFSFP-recovered/
```

The following would be recovered in the local directory:

1. */tmp/HDFSFP-recovered/user/root/employee* - The files and sub-directories present in the HDFS location */user/root/employee* are recovered in cleartext form.
2. */tmp/HDFSFP-recovered/user/ptyitusr/prot/employee* - The files and sub-directories present in the HDFS location */user/ptyitusr/prot/employee* are recovered in cleartext form.

➤ **To recover the protected data from a Hive warehouse directory to a local file system directory:**

1. Execute the following command to retrieve the protected data from a Hive warehouse directory.
   <PROTEGRITY_DIR>/hdfsfp/recover.sh –srcpath <protected HDFS path to be unprotected> -destpath <destination directory in the local file system>
   The cleartext data from the protected HDFS path is stored in the destination directory.
2. If you need to ensure that the existing Hive queries for the table function, then perform the following steps.
   a) Execute the following command to delete the warehouse directory for the table.
      hadoop fs –rm –r <hive.metastore.warehouse.dir>/tablename
   b) Move the destination directory with the cleartext data in HDFS using the following command.
      hadoop fs –put <destination directory in the local file system>/user/hive/warehouse/table_name <hive.metastore.warehouse.dir>/tablename
   c) To view the cleartext data in the table, use the following command.
      Select * from tablename

3.4 Uninstalling Big Data Protector from a Cluster

This section describes the procedure for uninstalling the Big Data Protector from the cluster.

3.4.1 Verifying the Prerequisites for Uninstalling Big Data Protector

If you are configuring the Big Data Protector with a Kerberos-enabled Hadoop cluster, then ensure that the HDFS superuser (*hdfs*) has a valid Kerberos ticket.

3.4.2 Removing the Cluster from the ESA

Before uninstalling Big Data Protector from the cluster, the cluster should be deleted from the ESA.

For more information about deleting the cluster from the ESA, refer to section *5.14.3 Removing a Cluster*.

3.4.3 Uninstalling Big Data Protector from the Cluster

Depending on the requirements, perform the following tasks to uninstall the Big Data Protector from the cluster.

3.4.3.1 Removing HDFSFP Configuration for Yarn (MRv2)

If HDFSFP is configured for Yarn (MRv2), then the configuration should be removed before uninstalling Big Data Protector.

➤ **To remove HDFSFP configuration for Yarn (MRv2) after uninstalling Big Data Protector:**

1. Remove the **com.protegrity.hadoop.fileprotector.crypto.codec.PtyCryptoCodec** codec from the *io.compression.codecs* property in the *core-site.xml* file.
2. Modify the value of the *mapreduce.output.fileoutputformat.compress* property in the *mapred-site.xml* file to *false*.

3. Remove the value of the *mapreduce.output.fileoutputformat.compress.codec* property in the *mapred-site.xml* file.
4. Remove the *<PROTEGRITY_DIR>/hdfsfp/** path from the *yarn.application.classpath* property in the *yarn-site.xml* file.
5. Restart the HDFS and Yarn services.

3.4.3.2 Removing HDFSFP Configuration for MapReduce, v1 (MRv1)

If HDFSFP is configured for MapReduce, v1 (MRv1), then the configuration should be removed before uninstalling Big Data Protector.

➤ **To remove HDFSFP configuration for MRv1 after uninstalling Big Data Protector:**

1. Remove the **com.protegrity.hadoop.fileprotector.crypto.codec.PtyCryptoCodec** codec from the *io.compression.codecs* property in the *core-site.xml* file.
2. Modify the value of the *mapred.output.compress* property in the *mapred-site.xml* file to *false*.
3. Remove the value of the *mapred.output.compression.codec* property in the *mapred-site.xml* file.
4. Restart the HDFS and MapReduce services.

3.4.3.3 Removing Configuration for Hive Protector if HDFSFP is not Installed

If the Hive protector is used and HDFSFP is not installed, then the configuration should be removed before uninstalling Big Data Protector.

➤ **To remove configuration for Hive protector if HDFSFP is not installed:**

1. If you are using a Hadoop distribution with a Management UI, then remove the value *com.protegrity.hive.PtyHiveUserPreHook* from the *hive.exec.pre.hooks* property, from the *hive-site.xml* file using the configuration management UI.
2. If you are using a Hadoop distribution without a Management UI, then remove the following property in the *hive-site.xml* file from all nodes.

```
<property>
<name>hive.exec.pre.hooks</name>
<value>hive.exec.pre.hooks=com.protegrity.hive.PtyHiveUserPreHook</value>
</property>
```

3.4.3.4 Removing Configurations for Hive Support in HDFSFP

If Hive is used with HDFSFP, then the configuration should be removed before uninstalling Big Data Protector.

➤ **To remove configurations for Hive support in HDFSFP:**

1. If you are using a Hadoop distribution with a Management UI, then perform the following steps.
    a) In the *hive-site.xml* file, remove the value of the *mapreduce.job.maps* property, using the Management UI.
    b) In the *hive-site.xml* file, remove the value *com.protegrity.hadoop.fileprotector.hive.PtyHivePreHook* from the *hive.exec.pre.hooks* property, using the configuration management UI.
2. If you are using a Hadoop distribution without a Management UI, then perform the following steps.
    a) Remove the following property in the *hive-site.xml* file on all nodes.

```
<property>
<name>mapreduce.job.maps</name>
```

```
<value>1</value>
</property>
```

b) Remove the following property in the *hive-site.xml* file on all nodes.

```
<property>
<name>hive.exec.pre.hooks</name>
<value>com.protegrity.hadoop.fileprotector.hive.PtyHivePreHook</value>
</property>
```

3.4.3.5 Removing the Configuration Properties when HDFSFP is not Installed

If you are using HDP, version 2.2 or higher (Hortonworks), or PHD, version 3.0 or higher (Pivotal Hadoop), and you have not installed HDFSFP, then the configuration should be removed before uninstalling Big Data Protector.

➤ **To remove the configuration properties:**

1. Remove the following entries from the *mapreduce.application.classpath* property in the *mapred-site.xml* file.

   *<PROTEGRITY_DIR>/pepmapreduce/lib/**

   *<PROTEGRITY_DIR>/pephive/lib/**

   *<PROTEGRITY_DIR>/peppig/lib/**

2. Remove the following entries from the *yarn.application.classpath* property in the *yarn-site.xml* file.

   *<PROTEGRITY_DIR>/pepmapreduce/lib/**

   *<PROTEGRITY_DIR>/pephive/lib/**

   *<PROTEGRITY_DIR>/peppig/lib/**

3. Restart the Yarn service.
4. Restart the MRv2 service.
5. Remove the following entries from the *tez.cluster.additional.classpath.prefix* property in the *tez-site.xml* file.

   *<PROTEGRITY_DIR>/pepmapreduce/lib/**

   *<PROTEGRITY_DIR>/pephive/lib/**

   *<PROTEGRITY_DIR>/peppig/lib/**

6. Restart the Tez services.

3.4.3.6 Removing HBase Configuration

If HBase is configured, then the configuration should be removed before uninstalling Big Data Protector.

➤ **To remove HBase configuration:**

1. If you are using a Hadoop distribution that has a Management UI, then remove the following HBase coprocessor region classes property value from the *hbase-site.xml* file in all the respective region server groups, using the Management UI.

   ```
   com.protegrity.hbase.PTYRegionObserver
   ```

2. If you are using a Hadoop distribution without a Management UI, then remove the following property in the *hbase-site.xml* file from all region server nodes.

   ```
   <property>
   <name>hbase.coprocessor.region.classes</name>
   <value>com.protegrity.hbase.PTYRegionObserver</value>
   ```

```
</property>
```

3. Restart all HBase services.

3.4.3.7 Removing the Defined Impala UDFs

If Impala is configured, then the defined Protegrity UDFs for the Impala protector should be removed before uninstalling Big Data Protector.

➤ To remove the defined Impala UDFs:

If you are not using a Kerberos-enabled Hadoop cluster, then run the following command to remove the defined Protegrity UDFs for the Impala protector using the *dropobjects.sql* script.

```
impala-shell -i <IP address of any Impala slave node> -f
<PROTEGRITY_DIR>/pepimpala/sqlscripts/dropobjects.sql
```

If you are using a Kerberos-enabled Hadoop cluster, then run the following command to remove the defined Protegrity UDFs for the Impala protector using the *dropobjects.sql* script.

```
impala-shell -i <IP address of any Impala slave node> -f
<PROTEGRITY_DIR>/pepimpala/sqlscripts/dropobjects.sql -k
```

3.4.3.8 Removing the Defined HAWQ UDFs

If HAWQ is configured, then the defined Protegrity UDFs for the HAWQ protector should be removed before uninstalling Big Data Protector.

➤ To remove the defined HAWQ UDFs:

Run the following command to remove the defined Protegrity UDFs for the HAWQ protector using the *dropobjects.sql* script.

```
psql -h <HAWQ Master Hostname> -p 5432 -f
<PROTEGRITY_DIR>/pephawq/sqlscripts/dropobjects.sql
```

3.4.3.9 Removing the Spark Protector Configuration

If the Spark protector is used, then the required configuration settings should be removed before uninstalling the Big Data Protector.

➤ To remove the Spark protector configuration:

1. Ensure that the Hadoop cluster is installed, configured, and running.
2. Update the *spark-defaults.conf* file to remove the following classpath entries, using Hadoop services, Cloudera Manager for Cloudera distributions, or Ambari Server for Hortonworks or Pivotal distributions, depending on the environment.
   ```
   spark.driver.extraClassPath=<PROTEGRITY_DIR>/pepspark/lib/*
   spark.executor.extraClassPath=<PROTEGRITY_DIR>/pepspark/lib/*
   ```
3. If HDFSFP is installed, then update the *spark-defaults.conf* file to remove the following classpath entries.
   ```
   spark.driver.extraClassPath=<PROTEGRITY_DIR>/pepspark/lib/*:<PROTEGRITY_DIR>/
   hdfsfp/*
   spark.executor.extraClassPath=<PROTEGRITY_DIR>/pepspark/lib/*:<PROTEGRITY_DIR
   >/hdfsfp/*
   ```
4. Save the *spark-defaults.conf* file.
5. Deploy the configuration change to all the nodes in the Hadoop cluster.
6. Restart the Spark services.

If Spark SQL is configured to run Hive UDFs, then the required configuration settings should be removed before uninstalling the Big Data Protector.

➤ **To remove the Spark SQL configuration:**

1. Ensure that the Hadoop cluster is installed, configured, and running.
2. Update the *spark-defaults.conf* file to remove the following classpath entries, using Hadoop services, Cloudera Manager for Cloudera distributions, or Ambari Server for Hortonworks or Pivotal distributions, depending on the environment.
   spark.driver.extraClassPath=<PROTEGRITY_DIR>/pephive/lib/*:<PROTEGRITY_DIR>/pepspark/lib/*
   spark.executor.extraClassPath=<PROTEGRITY_DIR>/pephive/lib/*:<PROTEGRITY_DIR>/pepspark/lib/*
3. If HDFSFP is installed, then update the *spark-defaults.conf* file to remove the following classpath entries.
   spark.driver.extraClassPath=<PROTEGRITY_DIR>/pephive/lib/*:<PROTEGRITY_DIR>/pepspark/lib/*:<PROTEGRITY_DIR>/hdfsfp/*
   spark.executor.extraClassPath=<PROTEGRITY_DIR>/pephive/lib/*:<PROTEGRITY_DIR>/pepspark/lib/*:<PROTEGRITY_DIR>/hdfsfp/*
4. Save the *spark-defaults.conf* file.
5. Deploy the configuration change to all the nodes in the Hadoop cluster.
6. Restart the Spark services.

3.4.3.10 Running the Uninstallation Script

➤ **To run the scripts for uninstalling the Big Data Protector on all nodes in the cluster:**

1. Login as the *sudoer* user and navigate to the *<PROTEGRITY_DIR>/cluster_utils* directory on the Lead node.
2. Run the following script to stop the PEP servers on all the nodes in the cluster.
   ```
   ./cluster_pepsrvctl.sh
   ```
3. Run the *uninstall.sh* utility.
   A prompt to confirm or cancel the Big Data Protector uninstallation appears.
4. Type *yes* to continue with the uninstallation.
5. When prompted, enter the *sudoer* password.
   The uninstallation script continues with the uninstallation of Big Data Protector.
   If you are using a Cloudera or MapR distribution, then the presence of an HDFS connection and a valid Kerberos ticket is also verified.

   The *<PROTEGRITY_DIR>/cluster_utils* directory continues to exist on the Lead node.

   This directory is retained to perform a cleanup in the event of the uninstallation failing on some nodes, due to unavoidable reasons, such as *host being down*.
6. After Big Data Protector is successfully uninstalled from all nodes, manually delete the *<PROTEGRITY_DIR>* directory from the Lead node.
7. If the *<PROTEGRITY_DIR>/defiance_dps_old* directory is present on any of the nodes in the cluster, then it can be manually deleted from the respective nodes.
8. Restart all Hadoop services.

4 Hadoop Application Protector

4.1 Using the Hadoop Application Protector

Various jobs written in the Hadoop cluster require data fields to be stored and retrieved. This data requires protection when it is at rest. The Hadoop Application Protector provides MapReduce, Hive and Pig the power to protect data while it is being processed and stored. Application programmers using these tools can include Protegrity software in their jobs to secure data.

For more information about using the protector APIs in various Hadoop applications and samples, refer to the following sections.

4.2 Prerequisites

Ensure that the following prerequisites are met before using Hadoop Application Protector:

- The Big Data Protector is installed and configured in the Hadoop cluster.
- The security officer has created the necessary security policy which creates data elements and user roles with appropriate permissions.

  For more information about creating security policies, data elements and user roles, refer to *Protection Enforcement Point Servers Installation Guide 6.6.5* and *Enterprise Security Administrator Guide 6.6.5*.
- The policy is deployed across the cluster.

For more information about the list of all APIs available to Hadoop applications, refer to sections *4.4 MapReduce APIs*, *4.5 Hive UDFs,* and *4.6 Pig UDFs*.

4.3 Samples

To run the samples provided with the Big Data Protector, the pre-packaged policy should be deployed from the ESA. During installation, specify the *INSTALL_DEMO* parameter as *Yes* in the *BDP.config* file.

The commands in the samples may require Hadoop-super-user permissions.

For more information about the samples, refer to section *11 Appendix: Samples*.

4.4 MapReduce APIs

This section describes the MapReduce APIs available for protection and unprotection in the Big Data Protector to build secure Big Data applications.

The Protegrity MapReduce protector only supports *bytes* converted from the *string* data type.

If *int*, *short*, or *long* format data is directly converted to *bytes* and passed as input to the API that supports *byte* as input and provides *byte* as output, then data corruption might occur.

If you are using the Bulk APIs for the MapReduce protector, then the following two modes for error handling and return codes are available:

- Default mode: Starting with the Big Data Protector, version 6.6.4, the Bulk APIs in the MapReduce protector will return the detailed error and return codes instead of *0* for *failure* and *1* for *success*. In addition, the MapReduce jobs involving Bulk APIs will provide error codes instead of throwing exceptions.

For more information about the error codes for Big Data Protector, version 6.6.5, refer to *Table 11-2 PEP Log Return Codes* and *Table 11-3 PEP Result Codes* in section *11 Appendix: Return Codes*.

- Backward compatibility mode: If you need to continue using the error handling capabilities provided with Big Data Protector, version 6.6.3 or lower, that is *0* for *failure* and *1* for *success*, then you can set this mode.

4.4.1 openSession()

This method opens a new user session for protect and unprotect operations. It is a good practice to create one session per user thread.

public synchronized int openSession(String parameter)

Parameters

`parameter`: An internal API requirement that should be set to 0.

Result

`1`: If session is successfully created

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus = mapReduceProtector.openSession("0");
```

Exception (and Error Codes)

ptyMapRedProtectorException: if session creation fails

4.4.2 closeSession()

This function closes the current open user session. Every instance of *ptyMapReduceProtector* opens only one session, and a session ID is not required to close it.

public synchronized int closeSession()

Parameters

None

Result

`1`: If session is successfully closed

`0`: If session closure is a failure

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus  = mapReduceProtector.openSession("0");
int closeSessionStatus = mapReduceProtector.closeSession();
```

Exception (and Error Codes)

None

4.4.3 getVersion()

This function returns the current version of the MapReduce protector.

public java.lang.String getVersion()

Parameters

None

Result

This function returns the current version of MapReduce protector.

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus = mapReduceProtector.openSession("0");
String version = mapReduceProtector.getVersion();
int closeSessionStatus = mapReduceProtector.closeSession();
```

4.4.4 getCurrentKeyId()

This method returns the current Key ID for the data element which contains the *KEY ID* attribute, while creating the data element, such as ASE-256, ASE-128, and so on.

public int getCurrentKeyId(java.lang.String dataElement)

Parameters

`dataElement`: Name of the data element

Result

This method returns the current Key ID for the data element containing the *KEY ID* attribute.

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus = mapReduceProtector.openSession("0");
int currentKeyId = mapReduceProtector.getCurrentKeyId("ENCRYPTION_DE");
int closeSessionStatus = mapReduceProtector.closeSession();
```

4.4.5 checkAccess()

This method checks the access of the user for the specified data element.

public boolean checkAccess(java.lang.String dataElement, byte bAccessType)

Parameters

`dataElement`: Name of the data element

`bAccessType`: Type of the access of the user for the data element.

The following are the different values for the bAccessType variable:

```
DELETE      0x01
PROTECT     0x02
REPROTECT   0x04
UNPROTECT   0x08
CREATE      0x10
MANAGE      0x20
```

Result

`1`: If the user has access to the data element

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus = mapReduceProtector.openSession("0");
byte bAccessType = 0x02;
boolean isAccess = mapReduceProtector.checkAccess("DE_PROTECT" , bAccessType );
int closeSessionStatus = mapReduceProtector.closeSession();
```

4.4.6 getDefaultDataElement()

This method returns default data element configured in security policy.

public String getDefaultDataElement(String policyName)

Parameters

`policyName`: Name of policy configured using Policy management in ESA.

Result

Default data element name configured in a given policy.

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus    = mapReduceProtector.openSession("0");
String defaultDataElement = mapReduceProtector.getDefaultDataElement("my_policy");
int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: If unable to return default data element name

4.4.7 protect()

Protects the data provided as a byte *array*. The type of protection applied is defined by *dataElement*.

public byte[] protect(String dataElement, byte[] data)

Parameters

`dataElement`: Name of the data element to be protected

`data`: Byte *array* of data to be protected

The Protegrity MapReduce protector only supports *bytes* converted from the *string* data type.

If *int*, *short*, or *long* format data is directly converted to *bytes* and passed as input to the API that supports *byte* as input and provides *byte* as output, then data corruption might occur.

If you are using the *Protect* API which accepts *byte* as input and provides *byte* as output, then ensure that when unprotecting the data, the *Unprotect* API, with *byte* as input and *byte* as output is utilized. In addition, ensure that the *byte* data being provided as input to the Protect API has been converted from a *string* data type only.

Result

Byte array of protected data

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus = mapReduceProtector.openSession("0");
byte[] bResult        = mapReduceProtector.protect(
                          "DE_PROTECT","protegrity".getBytes());
int closeSessionStatus = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: If unable to protect data

4.4.8 protect()

Protects the data provided as *int*. The type of protection applied is defined by *dataElement*.

public int protect(String dataElement, int data)

Parameters

dataElement: Name of the data element to be protected

data: *int* to be protected

Result

Protected *int* data

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus = mapReduceProtector.openSession("0");
int bResult           = mapReduceProtector.protect(
                          "DE_PROTECT",1234);
int closeSessionStatus = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: If unable to protect data

4.4.9 protect()

Protects the data provided as *long*. The type of protection applied is defined by *dataElement*.

public long protect(String dataElement, long data)

Parameters

dataElement: Name of the data element to be protected

data: *long* data to be protected

Result

Protected *long* data

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus = mapReduceProtector.openSession("0");
long  bResult           = mapReduceProtector.protect(
                          "DE_PROTECT",123412341234);
int closeSessionStatus = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: If unable to protect data

4.4.10 unprotect()

This function returns the data in its original form.

public byte[] unprotect(String dataElement, byte[] data)

Parameters

dataElement: Name of data element to be unprotected

data: *array* of data to be unprotected

The Protegrity MapReduce protector only supports *bytes* converted from the *string* data type.

If *int*, *short*, or *long* format data is directly converted to *bytes* and passed as input to the API that supports *byte* as input and provides *byte* as output, then data corruption might occur.

Result

Byte *array* of unprotected data

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus    = mapReduceProtector.openSession("0");
byte[] protectedResult   = mapReduceProtector.protect( "DE_PROTECT_UNPROTECT",
                             "protegrity".getBytes() );
byte[] unprotectedResult = mapReduceProtector.unprotect(
                             "DE_PROTECT_UNPROTECT", protectedResult );
int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: If unable to unprotect data

4.4.11 unprotect()

This function returns the data in its original form.

public int unprotect(String dataElement, int data)

Parameters

`dataElement:` Name of data element to be unprotected

`data:` *int* to be unprotected

Result

Unprotected *int* data

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus    = mapReduceProtector.openSession("0");
int protectedResult   = mapReduceProtector.protect( "DE_PROTECT_UNPROTECT",
                             1234 );
int unprotectedResult = mapReduceProtector.unprotect(
                             "DE_PROTECT_UNPROTECT", protectedResult );
int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: If unable to unprotect data

4.4.12 unprotect()

This function returns the data in its original form.

public long unprotect(String dataElement, long data)

Parameters

`dataElement:` Name of data element to be unprotected

`data:` *long* data to be unprotected

Result

Unprotected *long* data

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus    = mapReduceProtector.openSession("0");
long protectedResult   = mapReduceProtector.protect( "DE_PROTECT_UNPROTECT",
                            123412341234 );
long unprotectedResult = mapReduceProtector.unprotect(
                            "DE_PROTECT_UNPROTECT", protectedResult );
int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: If unable to unprotect data

4.4.13 bulkProtect()

This is used when a set of data needs to be protected in a bulk operation. It helps to improve performance.

public byte[][] bulkProtect(String dataElement, List <Integer> errorIndex, byte[][] inputDataItems)

Parameters

`dataElement`: Name of data element to be protected

`errorIndex`: *array* used to store all error indices encountered while protecting each data entry in *inputDataItems*

`inputDataItems`: Two-dimensional *array* to store bulk data for protection

Result

Two-dimensional byte array of protected data.

If the Backward Compatibility mode is not set, then the appropriate error code appears. For more information about the error codes, refer to *Table 11-2 PEP Log Return Codes* and *Table 11-3 PEP Result Codes* in section *11 Appendix: Return Codes*.

If the Backward Compatibility mode is set, then the Error Index includes one of the following values, per entry in the bulk protect operation:

- 1: The protect operation for the entry is successful.
- 0: The protect operation for the entry is unsuccessful.
  - For more information about the failed entry, view the logs available in ESA Forensics.
- Any other value or garbage return value: The protect operation for the entry is unsuccessful. For more information about the failed entry, view the logs available in ESA Forensics.

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
      int openSessionStatus    = mapReduceProtector.openSession("0");
      List<Integer> errorIndex = new ArrayList<Integer>();

      byte[][] protectData     = {"protegrity".getBytes{}, "protegrity".getBytes(),
                                  "protegrity".getBytes(), "protegrity".getBytes()};

      byte[][] protectedData = mapReduceProtector.bulkProtect( "DE_PROTECT",
                  errorIndex, protectData );

      System.out.print("Protected Data: ");
      for(int i = 0; i < protectedData.length; i++)
      {
```

```
      //THIS WILL PRINT THE PROTECTED DATA
          System.out.print(protectedData[i] == null ? null : new
       String(protectedData[i]));
          if(i < protectedData.length - 1)
          {
             System.out.print(",");
          }
       }

System.out.println("");
System.out.print("Error Index: ");
       for(int i = 0; i < errorIndex.size(); i++)
       {
            System.out.print(errorIndex.get( i ));
            if(i < errorIndex.size() - 1)
            {
              System.out.print(",");
            }
       }
//ABOVE CODE WILL PRINT THE ERROR INDEXES

int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: If an error is encountered during bulk protection of data

4.4.14 bulkProtect()

This is used when a set of data needs to be protected in a bulk operation. It helps to improve performance.

public int[] bulkProtect(String dataElement, List <Integer> errorIndex, int[] inputDataItems)

Parameters

`dataElement`: Name of data element to be protected

`errorIndex`: *array* used to store all error indices encountered while protecting each data entry in input Data Items

`inputDataItems`: *array* to store bulk *int* data for protection

Result

int array of protected data

If the Backward Compatibility mode is not set, then the appropriate error code appears. For more information about the error codes, refer to *Table 11-2 PEP Log Return Codes* and *Table 11-3 PEP Result Codes* in section *11 Appendix: Return Codes*.

If the Backward Compatibility mode is set, then the Error Index includes one of the following values, per entry in the bulk protect operation:

- 1: The protect operation for the entry is successful.
- 0: The protect operation for the entry is unsuccessful.
  - For more information about the failed entry, view the logs available in ESA Forensics.
- Any other value or garbage return value: The protect operation for the entry is unsuccessful. For more information about the failed entry, view the logs available in ESA Forensics.

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
       int openSessionStatus    = mapReduceProtector.openSession("0");
       List<Integer> errorIndex = new ArrayList<Integer>();

       int[] protectData     = {1234, 5678, 9012, 3456};

       int[] protectedData = mapReduceProtector.bulkProtect( "DE_PROTECT",
                   errorIndex, protectData );

//CHECK THE ERROR INDEXES FOR ERRORS
       System.out.print("Error Index: ");
       for(int i = 0; i < errorIndex.size(); i++)
       {
            System.out.print(errorIndex.get( i ));
            if(i < errorIndex.size() - 1)
            {
              System.out.print(",");
            }
       }
//ABOVE CODE WILL ONLY PRINT THE ERROR INDEXES

int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: If an error is encountered during bulk protection of data

4.4.15 bulkProtect()

This is used when a set of data needs to be protected in a bulk operation. It helps to improve performance.

public long[] bulkProtect(String dataElement, List <Integer> errorIndex, long[] inputDataItems)

Parameters

`dataElement`: Name of data element to be protected

`errorIndex`: array used to store all error indices encountered while protecting each data entry in input Data Items

`inputDataItems`: array to store bulk long data for protection

Result

Long array of protected data

If the Backward Compatibility mode is not set, then the appropriate error code appears. For more information about the error codes, refer to *Table 11-2 PEP Log Return Codes* and *Table 11-3 PEP Result Codes* in section *11 Appendix: Return Codes*.

If the Backward Compatibility mode is set, then the Error Index includes one of the following values, per entry in the bulk protect operation:

- 1: The protect operation for the entry is successful.
- 0: The protect operation for the entry is unsuccessful.
  - For more information about the failed entry, view the logs available in ESA Forensics.

- Any other value or garbage return value: The protect operation for the entry is unsuccessful. For more information about the failed entry, view the logs available in ESA Forensics.

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
       int openSessionStatus    = mapReduceProtector.openSession("0");
       List<Integer> errorIndex = new ArrayList<Integer>();

       long[] protectData     = {123412341234, 567856785678, 901290129012,
                           345634563456};

       long[] protectedData = mapReduceProtector.bulkProtect( "DE_PROTECT",
                   errorIndex, protectData );

//CHECK THE ERROR INDEXES FOR ERRORS

       System.out.print("Error Index: ");
       for(int i = 0; i < errorIndex.size(); i++)
       {
            System.out.print(errorIndex.get( i ));
            if(i < errorIndex.size() - 1)
            {
              System.out.print(",");
            }
       }
//ABOVE CODE WILL ONLY PRINT THE ERROR INDEXES

int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: If an error is encountered during bulk protection of data

4.4.16 bulkUnprotect()

This method unprotects in bulk the *inputDataItems* with the required data element.

public byte[][] bulkUnprotect(String dataElement, List<Integer> errorIndex, byte[][] inputDataItems)

Parameters

`String dataElement`: Name of data element to be unprotected

`int[] error index`: *array* of the error indices encountered while unprotecting each data entry in *inputDataItems*

`byte[][] inputDataItems`: two-dimensional *array* to help store bulk data to be unprotected

Result

Two-dimensional byte *array* of unprotected data

If the Backward Compatibility mode is not set, then the appropriate error code appears. For more information about the error codes, refer to *Table 11-2 PEP Log Return Codes* and *Table 11-3 PEP Result Codes* in section *11 Appendix: Return Codes*.

If the Backward Compatibility mode is set, then the Error Index includes one of the following values, per entry in the bulk unprotect operation:

- 1: The unprotect operation for the entry is successful.
- 0: The unprotect operation for the entry is unsuccessful.

    - For more information about the failed entry, view the logs available in ESA Forensics.
- Any other value or garbage return value: The unprotect operation for the entry is unsuccessful. For more information about the failed entry, view the logs available in ESA Forensics.

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
       int openSessionStatus    = mapReduceProtector.openSession("0");
       List<Integer> errorIndex = new ArrayList<Integer>();

       byte[][] protectData     = {"protegrity".getBytes{}, "protegrity".getBytes(),
                                   "protegrity".getBytes(), "protegrity".getBytes()};

       byte[][] protectedData = mapReduceProtector.bulkProtect( "DE_PROTECT",
                   errorIndex, protectData );

//THIS WILL PRINT THE UNPROTECTED DATA
       System.out.print("Protected Data: ");
       for(int i = 0; i < protectedData.length; i++)
       {
           System.out.print(protectedData[i] == null ? null : new
       String(protectedData[i]));
           if(i < protectedData.length - 1)
           {
              System.out.print(",");
           }
       }

//THIS WILL PRINT THE ERROR INDEX FOR PROTECT OPERATION
System.out.println("");
System.out.print("Error Index: ");
       for(int i = 0; i < errorIndex.size(); i++)
       {
            System.out.print(errorIndex.get( i ));
            if(i < errorIndex.size() - 1)
            {
              System.out.print(",");
            }
       }

byte[][] unprotectedData = mapReduceProtector.bulkUnprotect( "DE_PROTECT",
                   errorIndex, protectedData );


//THIS WILL PRINT THE PROTECTED DATA
       System.out.print("UnProtected Data: ");
       for(int i = 0; i < unprotectedData.length; i++)
       {
           System.out.print(unprotectedData[i] == null ? null : new
       String(unprotectedData[i]));
           if(i < unprotectedData.length - 1)
           {
              System.out.print(",");
           }
       }

//THIS WILL PRINT THE ERROR INDEX FOR UNPROTECT OPERATION
       System.out.println("");
```

```
System.out.print("Error Index: ");
      for(int i = 0; i < errorIndex.size(); i++)
      {
           System.out.print(errorIndex.get( i ));
           if(i < errorIndex.size() - 1)
           {
             System.out.print(",");
           }
      }

int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: For errors when unprotecting data

4.4.17 bulkUnprotect()

This method unprotects in bulk the *inputDataItems* with the required data element.

public int[] bulkUnprotect(String dataElement, List<Integer> errorIndex, int[] inputDataItems)

Parameters

`String dataElement`: Name of data element to be unprotected

`int[] error index`: *array* of the error indices encountered while unprotecting each data entry in inputDataItems

`int[] inputDataItems`: *int* array to be unprotected

Result

unprotected *int* array data

If the Backward Compatibility mode is not set, then the appropriate error code appears. For more information about the error codes, refer to *Table 11-2 PEP Log Return Codes* and *Table 11-3 PEP Result Codes* in section *11 Appendix: Return Codes*.

If the Backward Compatibility mode is set, then the Error Index includes one of the following values, per entry in the bulk unprotect operation:

- 1: The unprotect operation for the entry is successful.
- 0: The unprotect operation for the entry is unsuccessful.
  - For more information about the failed entry, view the logs available in ESA Forensics.
- Any other value or garbage return value: The unprotect operation for the entry is unsuccessful. For more information about the failed entry, view the logs available in ESA Forensics.

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
      int openSessionStatus    = mapReduceProtector.openSession("0");
      List<Integer> errorIndex = new ArrayList<Integer>();

      int[] protectData     = {1234, 5678,9012,3456 };

      int[] protectedData = mapReduceProtector.bulkProtect( "DE_PROTECT",
                  errorIndex, protectData );

//THIS WILL PRINT THE ERROR INDEX FOR PROTECT OPERATION
System.out.println("");
```

```
System.out.print("Error Index: ");
       for(int i = 0; i < errorIndex.size(); i++)
       {
            System.out.print(errorIndex.get( i ));
            if(i < errorIndex.size() - 1)
            {
              System.out.print(",");
            }
       }

int[] unprotectedData = mapReduceProtector.bulkUnprotect( "DE_PROTECT",
                  errorIndex, protectedData );

//THIS WILL PRINT THE ERROR INDEX FOR UNPROTECT OPERATION
       System.out.println("");
System.out.print("Error Index: ");
       for(int i = 0; i < errorIndex.size(); i++)
       {
            System.out.print(errorIndex.get( i ));
            if(i < errorIndex.size() - 1)
            {
              System.out.print(",");
            }
       }

int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: For errors when unprotecting data

4.4.18 bulkUnprotect()

This method unprotects in bulk the *inputDataItems* with the required data element.

public long[] bulkUnprotect(String dataElement, List<Integer> errorIndex, long[] inputDataItems)

Parameters

`String dataElement`: Name of data element to be unprotected

`int[] error index`: *array* of the error indices encountered while unprotecting each data entry in inputDataItems

`long[] inputDataItems`: long *array* to be unprotected

Result

Unprotected long array data

If the Backward Compatibility mode is not set, then the appropriate error code appears. For more information about the error codes, refer to *Table 11-2 PEP Log Return Codes* and *Table 11-3 PEP Result Codes* in section *11 Appendix: Return Codes*.

If the Backward Compatibility mode is set, then the Error Index includes one of the following values, per entry in the bulk unprotect operation:

- 1: The unprotect operation for the entry is successful.
- 0: The unprotect operation for the entry is unsuccessful.
  - For more information about the failed entry, view the logs available in ESA Forensics.

- Any other value or garbage return value: The unprotect operation for the entry is unsuccessful. For more information about the failed entry, view the logs available in ESA Forensics.

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus    = mapReduceProtector.openSession("0");
List<Integer> errorIndex = new ArrayList<Integer>();

long[] protectData     = { 123412341234, 567856785678,
                           901290129012, 345634563456 };

long[] protectedData = mapReduceProtector.bulkProtect( "DE_PROTECT",
            errorIndex, protectData );

//THIS WILL PRINT THE ERROR INDEX FOR PROTECT OPERATION
System.out.println("");
System.out.print("Error Index: ");
for(int i = 0; i < errorIndex.size(); i++)
{
     System.out.print(errorIndex.get( i ));
     if(i < errorIndex.size() - 1)
     {
       System.out.print(",");
     }
}

long[] unprotectedData = mapReduceProtector.bulkUnprotect( "DE_PROTECT",
            errorIndex, protectedData );

//THIS WILL PRINT THE ERROR INDEX FOR UNPROTECT OPERATION
System.out.println("");
System.out.print("Error Index: ");
for(int i = 0; i < errorIndex.size(); i++)
{
     System.out.print(errorIndex.get( i ));
     if(i < errorIndex.size() - 1)
     {
       System.out.print(",");
     }
}

int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: For errors when unprotecting data

4.4.19 reprotect()

Data that has been protected earlier is protected again with a separate data element.

public byte[] reprotect(String oldDataElement, String newDataElement, byte[] data)

Parameters

`String oldDataElement:` Name of data element with which data was protected earlier
`String newDataElement:` Name of new data element with which data is reprotected
`byte[] data:` *array* of data to be protected

Result
Byte array of reprotected data

Example
```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus    = mapReduceProtector.openSession("0");
byte[] protectedResult   = mapReduceProtector.protect( "DE_PROTECT_1",
                           "protegrity".getBytes() );
byte[] reprotectedResult = mapReduceProtector.reprotect( "DE_PROTECT_1",
                           "DE_PROTECT_2", protectedResult );
int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception
ptyMapRedProtectorException: For errors while reprotecting data

4.4.20 reprotect()

Data that has been protected earlier is protected again with a separate data element.

public int reprotect(String oldDataElement, String newDataElement, int data)

Parameters
`String oldDataElement`: Name of data element with which data was protected earlier
`String newDataElement`: Name of new data element with which data is reprotected
`int data`: *array* of data to be protected

Result
Reprotected *int* data

Example
```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus    = mapReduceProtector.openSession("0");
int protectedResult   = mapReduceProtector.protect( "DE_PROTECT_1",
                           1234 );
int reprotectedResult = mapReduceProtector.reprotect( "DE_PROTECT_1",
                           "DE_PROTECT_2", protectedResult );
int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception
ptyMapRedProtectorException: For errors while reprotecting data

4.4.21 reprotect()

Data that has been protected earlier is protected again with a separate data element.

public long reprotect(String oldDataElement, String newDataElement, long data)

Parameters
`String oldDataElement`: Name of data element with which data was protected earlier
`String newDataElement`: Name of new data element with which data is reprotected
`long data`: *array* of data to be protected

Result
Reprotected *long* data

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus    = mapReduceProtector.openSession("0");
long protectedResult   = mapReduceProtector.protect( "DE_PROTECT_1",
                           123412341234 );
int reprotectedResult = mapReduceProtector.reprotect( "DE_PROTECT_1",
                           "DE_PROTECT_2", protectedResult );
int closeSessionStatus   = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: For errors while reprotecting data

4.4.22 hmac()

This method performs data hashing using the HMAC operation on a single data item with a data element, which is associated with *hmac*. It returns *hmac* value of the given data with the given data element.

public byte[] hmac(String dataElement, byte[] data)

Parameters

`String dataElement`: Name of data element for HMAC

`byte[] data`: *array* of data for HMAC

Result

Byte *array* of HMAC data

Example

```
ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
int openSessionStatus = mapReduceProtector.openSession("0");
byte[] bResult = mapReduceProtector.hmac( "DE_HMAC",
                                          "protegrity".getBytes() );
int closeSessionStatus = mapReduceProtector.closeSession();
```

Exception

ptyMapRedProtectorException: If an error occurs while doing HMAC

4.5 Hive UDFs

This section describes all Hive User Defined Functions (UDFs) that are available for protection and unprotection in Big Data Protector to build secure Big Data applications.

If you are using Ranger or Sentry, then ensure that your policy provides *create* access permissions to the required UDFs.

4.5.1 ptyGetVersion()

This UDF returns the current version of PEP.

ptyGetVersion()

Parameters

None

Result

This UDF returns the current version of PEP.

Example

```
create temporary function ptyGetVersion AS 'com.protegrity.hive.udf.ptyGetVersion';

drop table if exists test_data_table;

create table test_data_table(val string) row format delimited fields terminated by ','
stored as textfile;

load data local inpath 'test_data.csv' OVERWRITE INTO TABLE test_data_table;

select ptyGetVersion() from test_data_table;
```

4.5.2 ptyWhoAmI()

This UDF returns the current logged in user.

ptyWhoAmI()

Parameters

None

Result

This UDF returns the current logged in user.

Example

```
create temporary function ptyWhoAmI AS 'com.protegrity.hive.udf.ptyWhoAmI';

drop table if exists test_data_table;

create table test_data_table(val string) row format delimited fields terminated by ','
stored as textfile;

load data local inpath 'test_data.csv' OVERWRITE INTO TABLE test_data_table;

select ptyWhoAmI() from test_data_table;
```

4.5.3 ptyProtectStr()

This UDF protects *string* values.

ptyProtectStr(String input, String dataElement)

Parameters

`String input:` *String* value to protect

`String dataElement:` Name of data element to protect *string* value

Result

This UDF returns protected *string* value.

Example

```
create temporary function ptyProtectStr AS 'com.protegrity.hive.udf.ptyProtectStr';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val string) row format delimited fields terminated by ','
stored as textfile;
```

```
LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

insert overwrite table test_data_table select trim(val) from temp_table;

select ptyProtectStr(val, 'Token_alpha') from test_data_table;
```

4.5.4 ptyUnprotectStr()

` This UDF unprotects the existing protected *string* value.

ptyUnprotectStr(String input, String dataElement)

Parameters

`String input:` Protected *string* value to unprotect
`String dataElement:` Name of data element to unprotect *string* value

Result

This UDF returns unprotected *string* value.

Example

```
create temporary function ptyProtectStr AS 'com.protegrity.hive.udf.ptyProtectStr';

create temporary function ptyUnprotectStr AS 'com.protegrity.hive.udf.ptyUnprotectStr';

drop table if exists test_data_table;
drop table if exists temp_table;
drop table if exists protected_data_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val string) row format delimited fields terminated by ','
stored as textfile;

create table protected_data_table(protectedValue string) row format delimited fields
terminated by ',' stored as textfile;

LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

insert overwrite table test_data_table select trim(val) from temp_table;

insert overwrite table protected_data_table select ptyProtectStr(val, 'Token_alpha') from
test_data_table;

select ptyUnprotectStr(protectedValue, 'Token_alpha') from protected_data_table;
```

4.5.5 ptyReprotect()

This UDF reprotects *string* format protected data, which was earlier protected using the *ptyProtectStr* UDF, with a different data element.

ptyReprotect(String input, String oldDataElement, String newDataElement)

Parameters

`String input:` *String* value to reprotect
`String oldDataElement:` Name of data element used to protect the data earlier
`String newDataElement:` Name of new data element to reprotect the data

Result

This UDF returns protected *string* value.

Example

```
create temporary function ptyProtectStr AS 'com.protegrity.hive.udf.ptyProtectStr';

create temporary function ptyReprotect AS 'com.protegrity.hive.udf.ptyReprotect';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val string) row format delimited fields terminated by ','
stored as textfile;

create table test_protected_data_table(val string) row format delimited fields terminated
by ',' stored as textfile;

LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

insert overwrite table test_data_table select trim(val) from temp_table;

insert overwrite table test_protected_data_table select ptyProtectStr(val, 'Token_alpha')
from test_data_table;

create table test_reprotected_data_table(val string) row format delimited fields
terminated by ',' stored as textfile;

insert overwrite table test_reprotected_data_table select ptyReprotect(val,
'Token_alpha', 'new_Token_alpha') from test_protected_data_table;
```

4.5.6 ptyProtectUnicode()

This UDF protects *string* (Unicode) values.

ptyProtectUnicode(String input, String dataElement)

Parameters

`String input:` *String* (Unicode) value to protect

`String dataElement:` Name of data element to protect *string* (Unicode) value

This UDF should be used only if you need to tokenize Unicode data in Hive, and migrate the tokenized data from Hive to a Teradata database and detokenize the data using the Protegrity Database Protector.

Ensure that you use this UDF with a Unicode tokenization data element only.

For more information about migrating tokenized Unicode data to a Teradata database, refer to section *15 Appendix: Migrating Tokenized Unicode Data from and to a Teradata Database*.

Result

This UDF returns protected *string* value.

Example

```
create temporary function ptyProtectUnicode AS
'com.protegrity.hive.udf.ptyProtectUnicode';

drop table if exists temp_table;
```

```
create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

select ptyProtectUnicode(val, 'Token_unicode') from temp_table;
```

4.5.7 ptyUnprotectUnicode()

This UDF unprotects the existing protected *string* value.

ptyUnprotectUnicode(String input, String dataElement)

Parameters

`String input:` Protected *string* value to unprotect

`String dataElement:` Name of data element to unprotect *string* value

This UDF should be used only if you need to tokenize Unicode data in Teradata using the Protegrity Database Protector, and migrate the tokenized data from a Teradata database to Hive and detokenize the data using the Protegrity Big Data Protector for Hive.

Ensure that you use this UDF with a Unicode tokenization data element only.

For more information about migrating tokenized Unicode data from a Teradata database, refer to section *15 Appendix: Migrating Tokenized Unicode Data from and to a Teradata Database*.

Result

This UDF returns unprotected *string* (Unicode) value.

Example

```
create temporary function ptyProtectUnicode AS
'com.protegrity.hive.udf.ptyProtectUnicode';

create temporary function ptyUnprotectUnicode AS
'com.protegrity.hive.udf.ptyUnprotectUnicode';

drop table if exists temp_table;
drop table if exists protected_data_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table protected_data_table(protectedValue string) row format delimited fields
terminated by ',' stored as textfile;

LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

insert overwrite table protected_data_table select ptyProtectUnicode(val,
'Token_unicode') from temp_table;
```

4.5.8 ptyReprotectUnicode()

This UDF reprotects *string* format protected data, which was protected earlier using the *ptyProtectUnicode* UDF, with a different data element.

ptyReprotectUnicode(String input, String oldDataElement, String newDataElement)

Parameters

`String input:` *String* (Unicode) value to reprotect
`String oldDataElement:` Name of data element used to protect the data earlier
`String newDataElement:` Name of new data element to reprotect the data

This UDF should be used only if you need to tokenize Unicode data in Hive, and migrate the tokenized data from Hive to a Teradata database and detokenize the data using the Protegrity Database Protector.

Ensure that you use this UDF with a Unicode tokenization data element only.

For more information about migrating tokenized Unicode data to a Teradata database, refer to section *15 Appendix: Migrating Tokenized Unicode Data from and to a Teradata Database*.

Result

This UDF returns protected *string* value.

Example

```
create temporary function ptyProtectUnicode AS
'com.protegrity.hive.udf.ptyProtectUnicode';

create temporary function ptyReprotectUnicode AS
'com.protegrity.hive.udf.ptyReprotectUnicode';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val string) row format delimited fields terminated by ','
stored as textfile;

create table test_protected_data_table(val string) row format delimited fields terminated
by ',' stored as textfile;

LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

insert overwrite table test_data_table select cast(trim(val)) from temp_table;

insert overwrite table test_protected_data_table select ptyProtectUnicode(val,
'Unicode_Token') from test_data_table;

create table test_reprotected_data_table(val string) row format delimited fields
terminated by ',' stored as textfile;

insert overwrite table test_reprotected_data_table select ptyReprotectUnicode(val,
'Unicode_Token',’new_Unicode_Token’) from test_data_table;
```

4.5.9 ptyProtectInt()

This UDF protects *integer* values.

ptyProtectInt(int input, String dataElement)

Parameters

`int input:` *Integer* value to protect

`String dataElement:` Name of data element to protect *integer* value

Result

This UDF returns protected *integer* value.

Example

```
create temporary function ptyProtectInt AS 'com.protegrity.hive.udf.ptyProtectInt';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val int) row format delimited fields terminated by ','
stored as textfile;

LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

insert overwrite table test_data_table select cast(trim(val) as int) from temp_table;

select ptyProtectInt(val, 'Token_numeric') from test_data_table;
```

4.5.10 ptyUnprotectInt()

` This UDF unprotects the existing protected *integer* value.

ptyUnprotectInt(int input, String dataElement)

Parameters

`int input:` Protected *integer* value to unprotect

`String dataElement:` Name of data element to unprotect *integer* value

Result

This UDF returns unprotected *integer* value.

Example

```
create temporary function ptyProtectInt AS 'com.protegrity.hive.udf.ptyProtectInt';

create temporary function ptyUnprotectInt AS 'com.protegrity.hive.udf.ptyUnprotectInt';

drop table if exists test_data_table;
drop table if exists temp_table;
drop table if exists protected_data_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val int) row format delimited fields terminated by ','
stored as textfile;

create table protected_data_table(protectedValue int) row format delimited fields
terminated by ',' stored as textfile;

LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

insert overwrite table test_data_table select cast(trim(val) as int) from temp_table;
```

```
insert overwrite table protected_data_table select ptyProtectInt(val, 'Token_numeric')
from test_data_table;

select ptyUnprotectInt(protectedValue, 'Token_numeric') from protected_data_table;
```

4.5.11 ptyReprotect()

This UDF reprotects *integer* format protected data with a different data element.

ptyReprotect(int input, String oldDataElement, String newDataElement)

Parameters

`int input:` *Integer* value to reprotect
`String oldDataElement:` Name of data element used to protect the data earlier
`String newDataElement:` Name of new data element to reprotect the data

Result

This UDF returns protected *integer* value.

Example

```
create temporary function ptyProtectInt AS 'com.protegrity.hive.udf.ptyProtectInt';

create temporary function ptyReprotect AS 'com.protegrity.hive.udf.ptyReprotect';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val int) row format delimited fields terminated by ',' stored as
textfile;

create table test_data_table(val int) row format delimited fields terminated by ','
stored as textfile;

create table test_protected_data_table(val int) row format delimited fields terminated by
',' stored as textfile;

LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

insert overwrite table test_data_table select cast(trim(val) as int) from temp_table;

insert overwrite table test_protected_data_table select ptyProtectInt(val,
'Token_Integer') from test_data_table;

create table test_reprotected_data_table(val int) row format delimited fields terminated
by ',' stored as textfile;

insert overwrite table test_reprotected_data_table select ptyReprotect(val,
'Token_Integer', 'new_Token_Integer') from test_protected_data_table;
```

4.5.12 ptyProtectFloat()

This UDF protects *float* value.

ptyProtectFloat(Float input, String dataElement)

Parameters

`Float input:` *Float* value to protect
`String dataElement:` Name of data element to unprotect *float* value

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

This UDF returns protected *float* value.

Example

```
create temporary function ptyProtectFloat as 'com.protegrity.hive.udf.ptyProtectFloat';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val float) row format delimited fields terminated by ','
stored as textfile;

load data local inpath 'test_data.csv' overwrite into table temp_table;

insert overwrite table test_data_table select cast(trim(val) as float) from temp_table;

select ptyProtectFloat(val, 'FLOAT_DE') from test_data_table;
```

4.5.13 ptyUnprotectFloat()

This UDF unprotects protected *float* value.

ptyUnprotectFloat(Float input, String dataElement)

Parameters

`Float input:` Protected *float* value to unprotect

`String dataElement:` Name of data element to unprotect *float* value

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

This UDF returns unprotected *float* value.

Example

```
create temporary function ptyProtectFloat as 'com.protegrity.hive.udf.ptyProtectFloat';

create temporary function ptyUnprotectFloat as
'com.protegrity.hive.udf.ptyUnprotectFloat';

drop table if exists test_data_table;
drop table if exists temp_table;
drop table if exists protected_data_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val float) row format delimited fields terminated by ','
stored as textfile;

create table protected_data_table(protectedValue float) row format delimited fields
terminated by ',' stored as textfile;

load data local inpath 'test_data.csv' overwrite into table temp_table;
```

```
insert overwrite table test_data_table select cast(trim(val) as float) from temp_table;

insert overwrite table protected_data_table select ptyProtectFloat(val, 'FLOAT_DE') from
test_data_table;

select ptyUnprotectFloat(protectedValue, 'FLOAT_DE') from protected_data_table;
```

4.5.14 ptyReprotect()

This UDF reprotects *float* format protected data with a different data element.

ptyReprotect(Float input, String oldDataElement, String newDataElement)

Parameters

`Float input:` *Float* value to reprotect
`String oldDataElement:` Name of data element used to protect the data earlier
`String newDataElement:` Name of new data element to reprotect the data

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

This UDF returns protected *float* value.

Example

```
create temporary function ptyProtectFloat AS 'com.protegrity.hive.udf.ptyProtectFloat';

create temporary function ptyReprotect AS 'com.protegrity.hive.udf.ptyReprotect';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val float) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val float) row format delimited fields terminated by ','
stored as textfile;

create table test_protected_data_table(val float) row format delimited fields terminated
by ',' stored as textfile;

LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

insert overwrite table test_data_table select cast(trim(val) as float) from temp_table;

insert overwrite table test_protected_data_table select ptyProtectFloat(val,
'NoEncryption') from test_data_table;

create table test_reprotected_data_table(val float) row format delimited fields
terminated by ',' stored as textfile;

insert overwrite table test_reprotected_data_table select ptyReprotect(val, '
NoEncryption’,’NoEncryption’) from test_protected_data_table;
```

4.5.15 ptyProtectDouble()

This UDF protects *double* value.

ptyProtectDouble(Double input, String dataElement)

Parameters

`Double input:` *Double* value to unprotect

`String dataElement:` Name of data element to unprotect *double* value

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

This UDF returns protected *double* value.

Example

```
create temporary function ptyProtectDouble as
'com.protegrity.hive.udf.ptyProtectDouble';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val double) row format delimited fields terminated by ','
stored as textfile;

load data local inpath 'test_data.csv' overwrite into table temp_table;

insert overwrite table test_data_table select cast(trim(val) as double) from temp_table;

select ptyProtectDouble(val, 'DOUBLE_DE') from test_data_table;
```

4.5.16 ptyUnprotectDouble()

This UDF unprotects protected *double* value.

ptyUnprotectDouble(Double input, String dataElement)

Parameters

`Double input:` *Double* value to unprotect

`String dataElement:` Name of data element to unprotect *double* value

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

This UDF returns unprotected *double* value.

Example

```
create temporary function ptyProtectDouble as
'com.protegrity.hive.udf.ptyProtectDouble';

create temporary function ptyUnprotectDouble as
'com.protegrity.hive.udf.ptyUnprotectDouble';

drop table if exists test_data_table;
drop table if exists temp_table;
drop table if exists protected_data_table;

create table temp_table(val double) row format delimited fields terminated by ',' stored
as textfile;
```

```
create table test_data_table(val double) row format delimited fields terminated by ','
stored as textfile;

create table protected_data_table(protectedValue double) row format delimited fields
terminated by ',' stored as textfile;

load data local inpath 'test_data.csv' overwrite into table temp_table;

insert overwrite table test_data_table select cast(trim(val) as double) from temp_table;

insert overwrite table protected_data_table select ptyProtectDouble(val, 'DOUBLE_DE')
from test_data_table;

select ptyUnprotectDouble(protectedValue, 'DOUBLE_DE') from protected_data_table;
```

4.5.17 ptyReprotect()

This UDF reprotects *double* format protected data with a different data element.

ptyReprotect(Double input, String oldDataElement, String newDataElement)

Parameters

`Double input:` *Double* value to reprotect
`String oldDataElement:` Name of data element used to protect the data earlier
`String newDataElement:` Name of new data element to reprotect the data

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

This UDF returns protected *double* value.

Example

```
create temporary function ptyProtectDouble AS 'com.protegrity.hive.udf.ptyProtectDouble';

create temporary function ptyReprotect AS 'com.protegrity.hive.udf.ptyReprotect';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val double) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val double) row format delimited fields terminated by ','
stored as textfile;

create table test_protected_data_table(val double) row format delimited fields terminated
by ',' stored as textfile;

LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

insert overwrite table test_data_table select cast(trim(val) as double) from temp_table;

insert overwrite table test_protected_data_table select ptyProtectDouble(val,
'NoEncryption') from test_data_table;

create table test_reprotected_data_table(val double) row format delimited fields
terminated by ',' stored as textfile;
```

```
insert overwrite table test_reprotected_data_table select ptyReprotect(val, '
NoEncryption','NoEncryption') from test_protected_data_table;
```

4.5.18 ptyProtectBigInt()

This UDF protects *BigInt* value.

ptyProtectBigInt(BigInt input, String dataElement)

Parameters

`BigInt input:` Value to protect

`String dataElement:` Name of data element to protect value

Result

This UDF returns protected *BigInteger* value.

Example

```
create temporary function ptyProtectBigInt as 'com.protegrity.hive.udf.ptyProtectBigInt';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val bigint) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val bigint) row format delimited fields terminated by ','
stored as textfile;

load data local inpath 'test_data.csv' overwrite into table temp_table;

insert overwrite table test_data_table select cast(trim(val) as bigint) from temp_table;

select ptyProtectBigInt(val, 'BIGINT_DE') from test_data_table;
```

4.5.19 ptyUnprotectBigInt()

This UDF unprotects protected *BigInt* value.

ptyUnprotectBigInt(BigInt input, String dataElement)

Parameters

`BigInt input:` Protected value to unprotect

`String dataElement:` Name of data element to unprotect value

Result

This UDF returns unprotected *BigInteger* value.

Example

```
create temporary function ptyProtectBigInt as 'com.protegrity.hive.udf.ptyProtectBigInt';

create temporary function ptyUnprotectBigInt as
'com.protegrity.hive.udf.ptyUnprotectBigInt';

drop table if exists test_data_table;
drop table if exists temp_table;
drop table if exists protected_data_table;

create table temp_table(val bigint) row format delimited fields terminated by ',' stored
as textfile;
```

```
create table test_data_table(val bigint) row format delimited fields terminated by ','
stored as textfile;

create table protected_data_table(protectedValue bigint) row format delimited fields
terminated by ',' stored as textfile;

load data local inpath 'test_data.csv' overwrite into table temp_table;

insert overwrite table test_data_table select cast(trim(val) as bigint) from temp_table;

insert overwrite table protected_data_table select ptyProtectBigInt(val, 'BIGINT_DE')
from test_data_table;

select ptyUnprotectBigInt(protectedValue, 'BIGINT_DE') from protected_data_table;
```

4.5.20 ptyReprotect()

This UDF reprotects *BigInt* format protected data with a different data element.

ptyReprotect(Bigint input, String oldDataElement, String newDataElement)

Parameters

`Bigint input:` *Bigint* value to reprotect
`String oldDataElement:` Name of data element used to protect the data earlier
`String newDataElement:` Name of new data element to reprotect the data

Result

This UDF returns protected *bigint* value.

Example

```
create temporary function ptyProtectBigInt AS 'com.protegrity.hive.udf.ptyProtectBigInt';

create temporary function ptyReprotect AS 'com.protegrity.hive.udf.ptyReprotect';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val bigint) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val bigint) row format delimited fields terminated by ','
stored as textfile;

create table test_protected_data_table(val bigint) row format delimited fields terminated
by ',' stored as textfile;

LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

insert overwrite table test_data_table select cast(trim(val) as bigint) from temp_table;

insert overwrite table test_protected_data_table select ptyProtectBigInt(val,
'Token_BigInteger') from test_data_table;

create table test_reprotected_data_table(val bigint) row format delimited fields
terminated by ',' stored as textfile;

insert overwrite table test_reprotected_data_table select ptyReprotect(val, '
'BIGINT_DE', 'new_BIGINT_DE') from test_protected_data_table;
```

4.5.21 ptyProtectDec()

This UDF protects *decimal* value.

This API works only with the CDH 4.3 distribution.

ptyProtectDec(Decimal input, String dataElement)

Parameters

`Decimal input:` *Decimal* value to protect

`String dataElement:` Name of data element to protect *decimal* value

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

This UDF returns protected *decimal* value.

Example

```
create temporary function ptyProtectDec as 'com.protegrity.hive.udf.ptyProtectDec';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val decimal) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val decimal) row format delimited fields terminated by ','
stored as textfile;

load data local inpath 'test_data.csv' overwrite into table temp_table;

insert overwrite table test_data_table select cast(trim(val) as decimal) from temp_table;

select ptyProtectDec(val, 'BIGDECIMAL_DE') from test_data_table;
```

4.5.22 ptyUnprotectDec()

This UDF unprotects protected *decimal* value.

This API works only with the CDH 4.3 distribution.

ptyUnprotectDec(Decimal input, String dataElement)

Parameters

`Decimal input:` Protected d*ecimal* value to unprotect

`String dataElement:` Name of data element to unprotect d*ecimal* value

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

This UDF returns unprotected *decimal* value.

Example

```
create temporary function ptyProtectDec as 'com.protegrity.hive.udf.ptyProtectDec';
```

```
create temporary function ptyUnprotectDec as 'com.protegrity.hive.udf.ptyUnprotectDec';

drop table if exists test_data_table;
drop table if exists temp_table;
drop table if exists protected_data_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val decimal) row format delimited fields terminated by ','
stored as textfile;

create table protected_data_table(protectedValue decimal) row format delimited fields
terminated by ',' stored as textfile;

load data local inpath 'test_data.csv' overwrite into table temp_table;

insert overwrite table test_data_table select cast(trim(val) as decimal) from temp_table;

insert overwrite table protected_data_table select ptyProtectDec(val, 'BIGDECIMAL_DE')
from test_data_table;

select ptyUnprotectDec(protectedValue, 'BIGDECIMAL_DE') from protected_data_table;
```

4.5.23 ptyProtectHiveDecimal()

This UDF protects *decimal* value.

This API works only for distributions which include Hive, Version 0.11 and later.

ptyProtectHiveDecimal(Decimal input, String dataElement)

Parameters

`Decimal input:` *Decimal* value to protect

`String dataElement:` Name of data element to protect *decimal* value

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Before the *ptyProtectHiveDecimal()* UDF is called, Hive rounds off the decimal value in the table to 18 digits in scale, irrespective of the length of the data.

Result

This UDF returns protected *decimal* value.

Example

```
reate temporary function ptyProtectHiveDecimal as
'com.protegrity.hive.udf.ptyProtectHiveDecimal';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val decimal) row format delimited fields terminated by ','
stored as textfile;

load data local inpath 'test_data.csv' overwrite into table temp_table;
```

```
insert overwrite table test_data_table select cast(trim(val) as decimal) from temp_table;

select ptyProtectHiveDecimal(val, 'BIGDECIMAL_DE') from test_data_table;
```

4.5.24 ptyUnprotectHiveDecimal()

This UDF unprotects Decimal value.

This API works only for distributions which include Hive, Version 0.11 and later.

ptyUnprotectHiveDecimal(Decimal input, String dataElement)

Parameters

`Decimal input:` *Decimal* value to protect

`String dataElement:` Name of data element to unprotect *decimal* value

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Before the *ptyUnprotectHiveDecimal()* UDF is called, Hive rounds off the decimal value in the table to 18 digits in scale, irrespective of the length of the data.

Result

This UDF returns unprotected *decimal* value.

Example

```
create temporary function ptyProtectHiveDecimal as
'com.protegrity.hive.udf.ptyProtectHiveDecimal';

create temporary function ptyUnprotectHiveDecimal as
'com.protegrity.hive.udf.ptyUnprotectHiveDecimal';

drop table if exists test_data_table;
drop table if exists temp_table;
drop table if exists protected_data_table;

create table temp_table(val string) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val decimal) row format delimited fields terminated by ','
stored as textfile;

create table protected_data_table(protectedValue decimal) row format delimited fields
terminated by ',' stored as textfile;

load data local inpath 'test_data.csv' overwrite into table temp_table;

insert overwrite table test_data_table select cast(trim(val) as decimal) from temp_table;

insert overwrite table protected_data_table select ptyProtectHiveDecimal(val,
'BIGDECIMAL_DE') from test_data_table;

select ptyUnprotectHiveDecimal(protectedValue, 'BIGDECIMAL_DE') from
protected_data_table;
```

4.5.25 ptyReprotect()

This UDF reprotects *decimal* format protected data with a different data element.

This API works only for distributions which include Hive, Version 0.11 and later.

ptyReprotect(Decimal input, String oldDataElement, String newDataElement)

Parameters

`Decimal input:` *Decimal* value to reprotect
`String oldDataElement:` Name of data element used to protect the data earlier
`String newDataElement:` Name of new data element to reprotect the data

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

This UDF returns protected *decimal* value.

Example

```
create temporary function ptyProtectHiveDecimal AS
'com.protegrity.hive.udf.ptyProtectHiveDecimal';

create temporary function ptyReprotect AS 'com.protegrity.hive.udf.ptyReprotect';

drop table if exists test_data_table;
drop table if exists temp_table;

create table temp_table(val decimal) row format delimited fields terminated by ',' stored
as textfile;

create table test_data_table(val decimal) row format delimited fields terminated by ','
stored as textfile;

create table test_protected_data_table(val decimal) row format delimited fields
terminated by ',' stored as textfile;

LOAD DATA LOCAL INPATH 'test_data.csv' OVERWRITE INTO TABLE temp_table;

insert overwrite table test_data_table select cast(trim(val) as decimal) from temp_table;

insert overwrite table test_protected_data_table select ptyProtectHiveDecimal(val,
'NoEncryption') from test_data_table;

create table test_reprotected_data_table(val decimal) row format delimited fields
terminated by ',' stored as textfile;

insert overwrite table test_reprotected_data_table select ptyReprotect(val, '
NoEncryption’,’NoEncyption’) from test_protected_data_table;
```

4.6 Pig UDFs

This section describes all Pig UDFs that are available for protection and unprotection in Big Data Protector to build secure Big Data applications.

4.6.1 ptyGetVersion()

This UDF returns the current version of PEP.

ptyGetVersion()

Parameters

None

Result

chararray: Version number

Example

```
REGISTER /opt/protegrity/Hadoop-protector/lib/peppig-0.10.0.jar;
                                                    // register pep pig version
DEFINE ptyGetVersion com.protegrity.pig.udf.ptyGetVersion;
                                                    //define UDF
employees = LOAD ‘employee.csv’ using PigStorage(‘,’)
                  AS (eid:chararray,name:chararray, ssn:chararray);
                                                    // load employee.csv from HDFS path
version = FOREACH employees GENERATE ptyGetVersion();
DUMP version;
```

4.6.2 ptyWhoAmI()

This UDF returns the current logged in user name.

ptyWhoAmI()

Parameters

None

Result

chararray: User name

Example

```
REGISTER /opt/protegrity/Hadoop_protector/lib/peppig-0.10.0.jar;
DEFINE ptyWhoAmI com.protegrity.pig.udf.ptyWhoAmI;
employees = LOAD ‘employee.csv’ using PigStorage(‘,’)
            AS (eid:chararray, name:chararray, ssn:chararray);
username = FOREACH employees GENERATE ptyWhoAmI();
DUMP username;
```

4.6.3 ptyProtectInt()

This UDF returns protected value for *integer* data.

ptyProtectInt (int data, chararray dataElement)

Parameters

int data: Data to protect

chararray dataElement: Name of data element to use for protection

Result

Protected value for given numeric data

Example

```
REGISTER /opt/protegrity/hadoop_protector/lib/peppig-0.10.0.jar;
DEFINE ptyProtectInt com.protegrity.pig.udf.ptyProtectInt;
employees = LOAD ‘employee.csv’ using PigStorage(‘,’) AS (eid:int, name:chararray,
ssn:chararray);
data_p = FOREACH employees GENERATE ptyProtectInt(eid, ‘token_integer’);
DUMP data_p;
```

4.6.4 ptyUnprotectInt()

This UDF returns unprotected value for protected *integer* data.

ptyUnprotectInt (int data, chararray dataElement)

Parameters

`int data:` Protected data

`chararray dataElement:` Name of data element to use for unprotection

Result

Unprotected value for given protected *integer* data

Example

```
REGISTER /opt/protegrity/hadoop_protector/lib/peppig-0.10.0.jar;
DEFINE ptyProtectInt com.protegrity.pig.udf.ptyProtectInt;
DEFINE ptyUnprotectInt com.protegrity.pig.udf.ptyUnProtectInt;
employees = LOAD ‘employee.csv’ using PigStorage(‘,’) AS (eid:int, name:chararray,
ssn:chararray);
data_p = FOREACH employees GENERATE ptyProtectInt(eid, ‘token_integer’);
data_u = FOREACH data_p GENERATE ptyUnprotectInt(eid, ‘token_integer’);
DUMP data_u;
```

4.6.5 ptyProtectStr()

This UDF protects *string* value.

ptyProtectStr(chararray input, chararray dataElement)

Parameters

`chararray input:` *String* value to protect

`chararray dataElement:` Name of data element to unprotect *string* value

Result

`chararray`

Example

```
REGISTER /opt/protegrity/hadoop_protector/lib/peppig-0.10.0.jar;
DEFINE ptyProtectStr com.protegrity.pig.udf.ptyProtectStr;
employees = LOAD ‘employee.csv’ using PigStorage(‘,’) AS (eid:chararray, name:chararray,
ssn:chararray);
data_p = FOREACH employees GENERATE ptyProtectIntStr(name, ‘token_alphanumeric’);
DUMP data_p
```

4.6.6 ptyUnprotectStr()

This UDF unprotects protected *string* value.

ptyUnprotectStr (chararray input, chararray dataElement)

Parameters

`chararray input:` Unprotected *string* value

`chararray dataElement:` Name of data element to unprotect *string* value

Result

`chararray:` Unprotected value

Example

```
REGISTER /opt/protegrity/hadoop_protector/lib/peppig-0.10.0.jar;
DEFINE ptyProtectInt com.protegrity.pig.udf.ptyProtectStr;
DEFINE ptyUnprotectInt com.protegrity.pig.udf.ptyUnProtectStr;
employees = LOAD ‘employee.csv’ using PigStorage(‘,’) AS (eid:chararray, name:chararray,
ssn:chararray);
data_p = FOREACH employees
            GENERATE ptyProtectStr(name, ‘token_alphanumeric’) as name:chararray
DUMP data_p;
data_u = FOREACH data_p GENERATE ptyUnprotectStr(ssn, ‘Token_alphanumeric’);
DUMP data_u;
```

5 HDFS File Protector (HDFSFP)

5.1 Overview of HDFSFP

The files stored in HDFS are plain text files and can be accessed with a POSIX-based file system access control. These files may contain sensitive data which is vulnerable with exposure to unwanted users.

The HDFS File Protector (HDFSFP) helps to transparently protect these files as they are stored into HDFS and allow only authorized users to access the content in the files.

5.2 Features of HDFSFP

The following are the features of HDFSFP:

- Protects and stores files in HDFS and retrieves the protected files in the clear from HDFS, as per centrally defined security policy and access control.
- Stores and retrieves from HDFS transparently for the user, depending upon their access control rights.
- Preserves Hadoop distributed data processing ensuring that protected content is processed on data nodes independently.
- Blocks the addressing of files by the defined access control pass-through, transparently without any protection or unprotection.
- Protects temporary data, such as intermediate files generated by the MapReduce job.
- Provides recursive access control for HDFS directories and files. Protects directories, its subdirectories and files, as per defined security policy and access control.
- Protects files at rest so that unauthorized users can view only the protected content.
- Adds minimum overhead for data processing in HDFS.
- Can be accessed using the command shell and Java API.

5.3 Protector Usage

Files stored in HDFS are plain text files. Access controls for HDFS are implemented by using file-based permissions that follow the UNIX permissions model. These files may contain sensitive data, making it vulnerable when exposed to unwanted users. These files should be transparently protected as they are stored into HDFS and the content should be exposed only to authorized users.

The files are stored and retrieved from HDFS using Hadoop ecosystem products, such as file shell commands, MapReduce jobs, and so on.

Any user or application with write access to protected data at rest in HDFS can delete, update or move the protected data. Hence though the protected data can be lost, the data is not compromised as the user or application do not access the original data in the clear. Ensure that the Hadoop administrator assigns file permissions in HDFS cautiously.

5.4 File Recover Utility

The File Recover utility recovers the contents from a protected file.

For more information about the File Recover Utility, refer to section *3.4.3 Recover Utility*.

5.5 HDFSFP Commands

Hadoop provides shell commands for modifying and administering HDFS. HDFSFP extends the modification commands to control access to files and directories in HDFS.

The section describes the commands supported in HDFSFP.

5.5.1 copyFromLocal

This command ingests local data into HDFS.

```
hadoop ptyfs -copyFromLocal <local path of file to copy> <destination HDFS directory path>
```

Result

- If the destination directory path is protected and the user executing the command has permissions to create and protect, then the data is ingested in encrypted form.
- If the destination directory path is protected and the user does not have permissions to create and protect, then the copy operation fails.
- If the destination HDFS directory path is not protected, then the data is ingested in clear form.

5.5.2 put

This command ingests local data into HDFS.

```
hadoop ptyfs -put <local path of file to copy> <destination HDFS directory path>
```

Result

- If the destination HDFS directory path is protected and the user executing the command has permissions to create and protect, then the data is ingested in encrypted form.
- If the destination HDFS directory path is protected and the user does not have permissions to create and protect, then the copy operation fails.
- If the destination HDFS directory path is not protected, then the data is ingested in clear form.

5.5.3 copyToLocal

This command is used to copy an HDFS file to a local directory.

```
hadoop ptyfs -copyToLocal <HDFS file path to copy> <destination local directory >
```

Result

- If the source HDFS file is protected and the user has unprotect permissions, then the file is copied to the destination directory in clear form.
- If the source HDFS file is not protected, then the file is copied to the destination directory.
- If the HDFS file is protected the user does not have unprotect permissions, then the copy operation fails.

5.5.4 get

This command copies an HDFS file to a local directory.

```
hadoop ptyfs -get <HDFS file path to copy> <destination local directory>
```

Result

- If the source HDFS file is protected and the user has unprotect permissions, then the file is copied to the destination directory in clear form.
- If the source HDFS file is not protected, then the file is copied to the destination directory.
- If the HDFS file is protected the user does not have unprotect permissions, then the copy operation fails.

5.5.5 cp

This command copies a file from one HDFS directory to another HDFS directory.

```
hadoop ptyfs -cp <source HDFS file path> <destination HDFS directory path>
```

Result

- If the source HDFS file is protected and the user has unprotect permissions for the source HDFS file, the destination directory is protected, and the user has permissions to protect and create on the destination HDFS directory path, then the file gets copied in encrypted form.
- If the source HDFS file is protected and the user does not have permissions to unprotect, then the copy operation fails.
- If the destination directory is protected and the user does not have permissions to protect and create, then the copy operation fails.
- If the source HDFS file is unprotected and destination directory is protected and the user has permissions to protect or create on the destination HDFS directory path, then the file is copied in encrypted form.
- If the source HDFS file is protected and the user has permissions to unprotect for the source HDFS file and destination HDFS directory path is not protected, then the file is copied in clear form.
- If the source HDFS file and destination HDFS directory path are unprotected, then the command works similar to the default Hadoop file shell *-cp* command.

5.5.6 mkdir

This command creates a new directory in HDFS.

```
hadoop ptyfs -mkdir <new HDFS directory path>
```

Result

- If the new directory is protected and the user has permissions to create, then the new directory is created.
- If the new directory is not protected, then this command runs similar to the default HDFS file shell *-mkdir* command.

5.5.7 mv

This command moves an HDFS file from one HDFS directory to another HDFS directory.

```
hadoop ptyfs -mv <source HDFS file path> <destination HDFS directory path>
```

Result

- If the source HDFS file is protected and the user has unprotect and delete permissions and the destination directory is also protected with the user having permissions to protect and create on the destination HDFS directory path, then the file is moved to the destination directory in encrypted form.
- If the HDFS file is protected and the user does not have unprotect and delete permissions or the destination directory is protected and the user does not have permissions to protect and create, then the move operation fails.
- If the source HDFS file is unprotected, the destination directory is protected and the user has permissions to protect and create on the destination HDFS directory path, then the file is copied in encrypted form.
- If the source HDFS file is protected and the user has permissions to unprotect and the destination HDFS directory path is not protected, then the file is copied in clear form.
- If the source HDFS file and destination HDFS directory path are unprotected, then the command works similar to the default Hadoop file shell *-cp* command.

5.5.8 rm

This command deletes HDFS files.

```
hadoop ptyfs -rm <HDFS file paths to delete>
```

Result

- If the HDFS file is protected and the user has permissions to delete on the HDFS file path, then the file is deleted.
- If the HDFS file is protected and the user does not have permissions to delete on the HDFS file path, then the delete operation fails.
- If the HDFS file is not protected, then the command works similar to the default Hadoop file shell *-rm* command.

5.5.9 rmr

This command deletes an HDFS directory, its subdirectories and files.

```
hadoop ptyfs -rmr <HDFS directory path to delete>
```

Result

- If the HDFS directory path is protected and the user has permissions to delete on the HDFS directory path, then the directory and its contents are deleted.
- If the HDFS directory path is protected and the user does not have permissions to delete on the HDFS directory path, then the delete operation fails.
- If the HDFS directory path is not protected, then the command works as the default Hadoop rm recursive (hadoop fs *-rmr or hadoop* fs *–rm -r*) command.

5.6 Ingesting Files Securely

To ingest files into HDFS securely, use the *put* and *CopyFromLocal* commands.

For more information, refer to sections *5.5.2 put* and *5.5.1 copyFromLocal*.

If you need to ingest data to a protected ACL in HDFS using Sqoop, then use the *-D target.output.dir* parameter before any tool-specific arguments as described in the following command.

```
sqoop import -D target.output.dir="/tmp/src" --driver com.mysql.jdbc.Driver
--connect "jdbc:mysql://master.localdomain/test" --username root --table test --
target-dir /tmp/src -m 1
```

In addition, if you need to add additional data to any existing data, then use the *--append* parameter as described in the following command.

```
sqoop import -D target.output.dir="/tmp/src" --driver com.mysql.jdbc.Driver
--connect "jdbc:mysql://master.localdomain/test" --username root --table test --
target-dir /tmp/src -m 1 --append
```

5.7 Extracting Files Securely

To extract files from HDFS securely, use the *get* and *CopyToLocal* commands.

For more information, refer to sections *5.5.4 get* and *5.5.3 copyToLocal*.

5.8 HDFSFP Java API

Protegrity provides a Java API for working with files and directories using HDFSFP. The Java API for HDFSFP provides an alternate means of working with HDFSFP besides the HDFSFP shell commands, `hadoop ptyfs`, and enables you to integrate HDFSFP with Java applications.

The section describes the Java API commands supported in HDFSFP.

5.8.1 copy

This command copies a file from one HDFS directory to another HDFS directory.

```
copy(java.lang.String srcs, java.lang.String dst)
```

Parameters

`srcs`: HDFS file path

`dst`: HDFS file or directory path

Returns

`True`: If the operation is successful

`Exception`: If the operation fails

Exception (and Error Codes)

The API returns an exception (`com.protegrity.hadoop.fileprotector.fs.ProtectorException`) if any of the following conditions are met:

- Input is null.
- The path does not exist.
- The user does not have protect and write permissions on the destination path in case the destination path is protected, or the user does not have unprotect permission on the source path or both.

For more information on exceptions, refer to the Javadoc provided with the HDFSFP Java API. The Javadoc can be found in *<protegrity_base_directory>/protegrity/hdfsfp/doc* on the Data Ingestion Node.

Result

- If the source HDFS file is protected and the user has unprotect permission for the source HDFS file, the destination directory is protected, the ACL entry for the directory is activated, and the user has permissions to protect and create on the destination HDFS directory path, then the file gets copied in encrypted form.
- If the source HDFS file is protected and the user does not have permission to unprotect, then the copy operation fails.
- If the destination directory is protected and the user does not have permissions to protect and create, then the copy operation fails.
- If the source HDFS file is unprotected and destination directory is protected, the ACL entry for the directory is activated, and the user has permissions to protect or create on the destination HDFS directory path, then the file is copied in encrypted form.
- If the source HDFS file is protected and the user has permissions to unprotect for the source HDFS file and destination HDFS directory path is not protected, then the file is copied in clear form.

5.8.2 copyFromLocal

This command ingests local data into HDFS.

```
copyFromLocal(java.lang.String[] srcs, java.lang.String dst)
```

Parameters

`srcs`: Array of local file paths

`dst`: HDFS directory path

Returns

`True`: If the operation is successful

`Exception`: If the operation fails

Exception (and Error Codes)

The API returns an exception (`com.protegrity.hadoop.fileprotector.fs.ProtectorException`) if any of the following conditions are met:

- Input is null.
- The path does not exist.
- The user does not have protect and write permissions on the destination path if it is protected.

For more information on exceptions, refer to the Javadoc provided with the HDFSFP Java API. The Javadoc can be found in *<protegrity_base_directory>/protegrity/hdfsfp/doc* on the Data Ingestion Node.

Result

- If the destination directory path is protected, the ACL entry for the directory is activated, and the user executing the command has permissions to create and protect, then the data is ingested in encrypted form.
- If the destination directory path is protected and the user does not have permissions to create and protect, then the copy operation fails.
- If the destination HDFS directory path is not protected, then the data is ingested in clear form.

5.8.3 copyToLocal

This command is used to copy an HDFS file or directory to a local directory.

```
copyToLocal(java.lang.String srcs, java.lang.String dst)
```

Parameters

`srcs:` HDFS file or directory path

`dst:` Local directory or file path

Returns

`True:` If the operation is successful

`Exception:` If the operation fails

Exception (and Error Codes)

The API returns an exception (`com.protegrity.hadoop.fileprotector.fs.ProtectorException`) if any of the following conditions are met:

- Input is null.
- The path does not exist.
- The user does not have unprotect and read permissions on the source path if it is protected.

For more information on exceptions, refer to the Javadoc provided with the HDFSFP Java API. The Javadoc can be found in *<protegrity_base_directory>/protegrity/hdfsfp/doc* on the Data Ingestion Node.

Result

- If the source HDFS file is protected, the ACL entry for the directory is activated, and the user has unprotect permission, then the file is copied to the destination directory in clear form.
- If the source HDFS file is not protected, then the file is copied to the destination directory.
- If the HDFS file is protected the user does not have unprotect permissions, then the copy operation fails.

5.8.4 deleteFile

This command deletes files from HDFS.

```
deleteFile(java.lang.String srcf, boolean skipTrash)
```

Parameters

`srcf:` HDFS file path

`skipTrash:` Boolean value which decides if the file should be moved to trash. If the Boolean value is true, then the file is not moved to trash;
if false, then the file is moved to trash.

Returns

`True:` If the operation is successful

`Exception:` If the operation fails

Exception (and Error Codes)

The API returns an exception (`com.protegrity.hadoop.fileprotector.fs.ProtectorException`) if any of the following conditions are met:

- Input is null.
- The path does not exist.
- The user does not have delete permission to the path.

For more information on exceptions, refer to the Javadoc provided with the HDFSFP Java API. The Javadoc can be found in *<protegrity_base_directory>/protegrity/hdfsfp/doc* on the Data Ingestion Node.

Result

- If the HDFS file is protected and the user has permission to delete on the HDFS file path, then the file is deleted.
- If the HDFS file is protected and the user does not have permission to delete on the HDFS file path, then the delete operation fails.

5.8.5 deleteDir

This command deletes recursively an HDFS directory, its subdirectories, and files.

```
deleteDir(java.lang.String srcdir, boolean skipTrash)
```

Parameters

`srcdir:` HDFS directory path

`skipTrash:` Boolean value which decides if the file should be moved to trash. If the Boolean value is true, then the directory is recursively not moved to trash; if false, then the directory is recursively moved to trash.

Returns

`True:` If the operation is successful

`Exception:` If the operation fails

Exception (and Error Codes)

The API returns an exception (`com.protegrity.hadoop.fileprotector.fs.ProtectorException`) if any of the following conditions are met:

- Input is null.
- The path does not exist.
- The user does not have delete permission to the path.

For more information on exceptions, refer to the Javadoc provided with the HDFSFP Java API. The Javadoc can be found in *<protegrity_base_directory/protegrity/hdfsfp/doc* on the Data Ingestion Node.

Result

- If the HDFS directory path is protected and the user has permission to delete on the HDFS directory path, then the directory and its contents are deleted.
- If the HDFS directory path is protected and the user does not have permission to delete on the HDFS directory path, then the delete operation fails.

5.8.6 mkdir

This command creates a new directory in HDFS.

```
mkdir(java.lang.String dir)
```

Parameters

`dir:` HDFS directory path

Returns

`True`: If the operation is successful
`Exception`: If the operation fails

Exception (and Error Codes)

The API returns an exception (`com.protegrity.hadoop.fileprotector.fs.ProtectorException`) if any of the following conditions are met:

- Input is null.
- The user does not have write permissions to the path.

For more information on exceptions, refer to the Javadoc provided with the HDFSFP Java API. The Javadoc can be found in *<protegrity_base_directory>/protegrity/hdfsfp/doc* on the Data Ingestion Node.

Result

- If the new directory path exists in ACL or the ACL path for the parent directory path is activated recursively, and the user has permissions to create, then the new directory with an activated ACL path is created.
- If the new directory path or its parent directory path is not present in ACL recursively, then the new directory is created without HDFSFP protection.

5.8.7 move

This command moves an HDFS file from one HDFS directory to another HDFS directory.

```
move(java.lang.String src, java.lang.String dst)
```

Parameters

`src`: HDFS file path
`dst`: HDFS file or directory path

Returns

`True`: If the operation is successful
`Exception`: If the operation fails

Exception (and Error Codes)

The API returns an exception (`com.protegrity.hadoop.fileprotector.fs.ProtectorException`) if any of the following conditions are met:

- Input is null.
- The path does not exist.
- The user does not have unprotect and read, or protect and write, or create permissions to the path.
- The user does not have protect and write permissions on the destination path in case the destination path is protected, or the user does not have unprotect permission on the source path or both.

For more information on exceptions, refer to the Javadoc provided with the HDFSFP Java API. The Javadoc can be found in *<protegrity_base_directory>/protegrity/hdfsfp/doc* on the Data Ingestion Node.

Result

- If the source HDFS file is protected, the ACL entry for the directory is activated, and the user has unprotect and delete permissions and the destination directory is also protected with the

user having permissions to protect and create on the destination HDFS directory path, then the file is moved to the destination directory in encrypted form.

- If the HDFS file is protected and the user does not have unprotect and delete permissions or the destination directory is protected and the user does not have permissions to protect and create, then the move operation fails.
- If the source HDFS file is unprotected, the destination directory is protected, the ACL entry for the directory is activated, and the user has permissions to protect and create on the destination HDFS directory path, then the file is copied in encrypted form.
- If the source HDFS file is protected and the user has permission to unprotect and the destination HDFS directory path is not protected, then the file is copied in clear form.

5.9 Developing Applications using HDFSFP Java API

This section describes the guidelines to follow when developing applications using the HDFSFP Java API.

The guidelines described in this section are a sample and assumes that */opt/protegrity* is the base installation directory of Big Data Protector. These guidelines would need to be modified based on your requirements.

5.9.1 Setting up the Development Environment

Ensure that the following steps are completed before you begin to develop applications using the HDFSFP Java API:

- Add the required HDFSFP Java API jar, *hdfsfp-x.x.x.jar,* in the Classpath.
- Instantiate the HDFSFP Java API function using the following command:

  ```
  PtyHdfsProtector protector = new PtyHdfsProtector();
  ```

  After successful instantiation, you are ready to call the HDFSFP Java API functions.

5.9.2 Protecting Data using the Class file

➤ **To protect data using the Class file:**

1. Compile the Java file to create a Class file with the following command.
   ```
   javac -cp .:<PROTEGRITY_DIR>/hdfsfp/hdfsfp-x.x.x.jar ProtectData.java -d .
   ```
2. Protect data using the Class file with the following command.
   ```
   hadoop ProtectData
   ```

5.9.3 Protecting Data using the JAR file

➤ **To protect data using the JAR file:**

1. Compile the Java file to create a Class file with the following command.
   ```
   javac -cp .:<PROTEGRITY_DIR>/hdfsfp/hdfsfp-x.x.x.jar ProtectData.java -d .
   ```
2. Create the JAR file from the Class file with the following command.
   ```
   jar -cvf protectData.jar ProtectData
   ```
3. Protect data using the JAR file with the following command.
   ```
   hadoop jar protectData.jar ProtectData
   ```

5.9.4 Sample Program for the HDFSFP Java API

```
public class ProtectData {

public static PtyHdfsProtector protector = new PtyHdfsProtector();
```

```
public void copyFromLocalTest(String[] srcs, String dstf)
{
	boolean result;

	try {
		result = protector.copyFromLocal(srcs, dstf);
	} catch (ProtectorException pe) {
		pe.printStackTrace();
	}
}
public void copyToLocalTest(String srcs, String dstf)
{
	boolean result;

	try {
		result = protector.copyToLocal(srcs, dstf);
	} catch (ProtectorException pe) {
		pe.printStackTrace();
	}
}
public void copyTest(String srcs, String dstf)
{
	boolean result;

	try {
		result = protector.copy(srcs, dstf);
	} catch (ProtectorException pe) {
		pe.printStackTrace();
	}
}
public void mkdirTest(String dir)
{
	boolean result;

	try {
		result = protector.mkdir(dir);
	} catch (ProtectorException pe) {
		pe.printStackTrace();
	}
}
public void moveTest(String srcs, String dstf)
{
	boolean result;

	try {
		result = protector.move(srcs,dstf);
	} catch (ProtectorException pe) {
		pe.printStackTrace();
	}
}
public void deleteFileTest(String file,boolean skipTrash)
{
	boolean result;

	try {
		result = protector.deleteFile(file, skipTrash);
	} catch (ProtectorException pe) {
		pe.printStackTrace();
	}
}
public void deleteDirTest(String dir,boolean skipTrash)
{
	boolean result;

	try {
		result = protector.deleteDir(dir, skipTrash);
```

```
        } catch (ProtectorException pe) {
              pe.printStackTrace();
        }
}
public static void main(String[] args) {
        ProtectData protect = new ProtectData();

        // Ingest Local Data into HDFS
        String srcsCFL[] = new String[2];
        srcsCFL[0] ="<Local source file location1>";
        srcsCFL[1] ="<Local Source file location2>";
        String dstfCFL ="<ACL activated HDFS destination directory location>";
        protect.copyFromLocalTest(srcsCFL, dstfCFL);

        // Extract HDFS file to Local
        String srcsCTL= "<ACL activated HDFS source file location>";
        String dstfCTL = "<Local destination directory location >";
        protect.copyToLocalTest(srcsCTL, dstfCTL);

        // Copy File from HDFS to HDFS
        String srcsCopy="<ACL activated HDFS source file location>";
        String dstfCopy ="<ACL activated HDFS destination directory location>";
        protect.copyTest(srcsCopy, dstfCopy);

        // Create HDFS Sub-Directory
        String dir = "<HDFS directory location>";
        protect.mkdirTest(dir);

        // Move from HDFS to HDFS
        String srcsMove = "<ACL activated HDFS source file location>";
        String dstfMove = "<ACL activated HDFS destination directory location>";
        protect.moveTest(srcsMove, dstfMove);

        // Delete File from HDFS
        String fileDelete = "<HDFS file location>";
        boolean skipTrashFile = false;
        protect.deleteFileTest(fileDelete,skipTrashFile);

        // Delete Sub-Directory and Children from HDFS
        String dirDelete = "<HDFS directory location>";
        boolean skipTrashDir = false;
        protect.deleteDirTest(dirDelete,skipTrashDir);
}
}
```

5.10 Quick Reference Tasks

This section provides a quick reference for the tasks that can be performed by users.

5.10.1 Protecting Existing Data

The *dfsadmin* utility protects existing data after creating ACL for the HDFS path. It is a two-step process. In first step, the user creates new ACL entries. In the second step, the user will activate the newly created ACL entries. After activation, all ACL entries will be protected automatically.

The steps for activating ACL entries can be done for single or multiple entries. After activating ACL entries, the HDFSFP infrastructure protects the HDFS path in the ACL entry.

While installing HDFSFP, you need to configure the ingestion user in the *BDP.config* file. The HDFS administrator would have to ensure that the ingestion user has full access to the directories that would be protected with HDFSFP. This user would be the authorized user for protection. Permissions to protect or create are configured in the security policy. After the *dfsadmin* utility activates ACL entry using the preconfigured ingested user, the HDFS File Protector protects the ACL path.

For more information about adding and activating an ACL entry, refer to sections *5.15.1 Adding an ACL Entry for Protecting Files or Folders* and *5.15.5 Activating Inactive ACL Entries*.

5.10.2 Reprotecting Files

For information about reprotecting files, refer to section *5.15.3 Reprotecting Files or Folders*.

5.11 Sample Demo Use Case

For information about the sample demo use cases, refer to *12 Appendix: HDFSFP Demo*.

HDFSFP can monitor policy and file or folder activity. This auditing can be done on a per policy basis.

The following event types can be audited when an event is generated on the action listed:

- Create or update ACL entry to protect or reprotect a file or folder
- Read/write an ACL encrypted file or folder
- Update or delete an ACL entry

Auditing qualifiers include success, failure, and auditing only when the user is audited for the same action.

5.12 Appliance components of HDFSFP

This section describes the active components, shipped with ESA and required to run HDFSFP.

5.12.1 Dfsdatastore Utility

This utility adds the Hadoop cluster under protection for HDFSFP.

5.12.2 Dfsadmin Utility

This utility handles management of access control entries for files and folders.

5.13 Access Control Rules for Files and Folders

Rules for files and folders stored or accessed in HDFS are managed by Access Control Lists (ACLs). The protection of HDFS files and folders is done after the ACL entry has been created. ACLs for multiple Hadoop clusters can be managed only from the ESA. Protegrity Cache is used to store or propagate secured ACLs across the clusters.

If you need to add, delete, search, update, or list a cluster, then use the DFS Cluster Management Utility (*dfsdatastore*).

If you need to protect, unprotect, reprotect, activate, search, or update ACLs, or get information about a job, then use the ACL Management Utility (*dfsadmin)*.

For more information about managing access control entries across clusters, refer to sections *5.14 Using the DFS Cluster Management Utility (*dfsdatastore) and *5.15 Using the ACL Management Utility (dfsadmin)*.

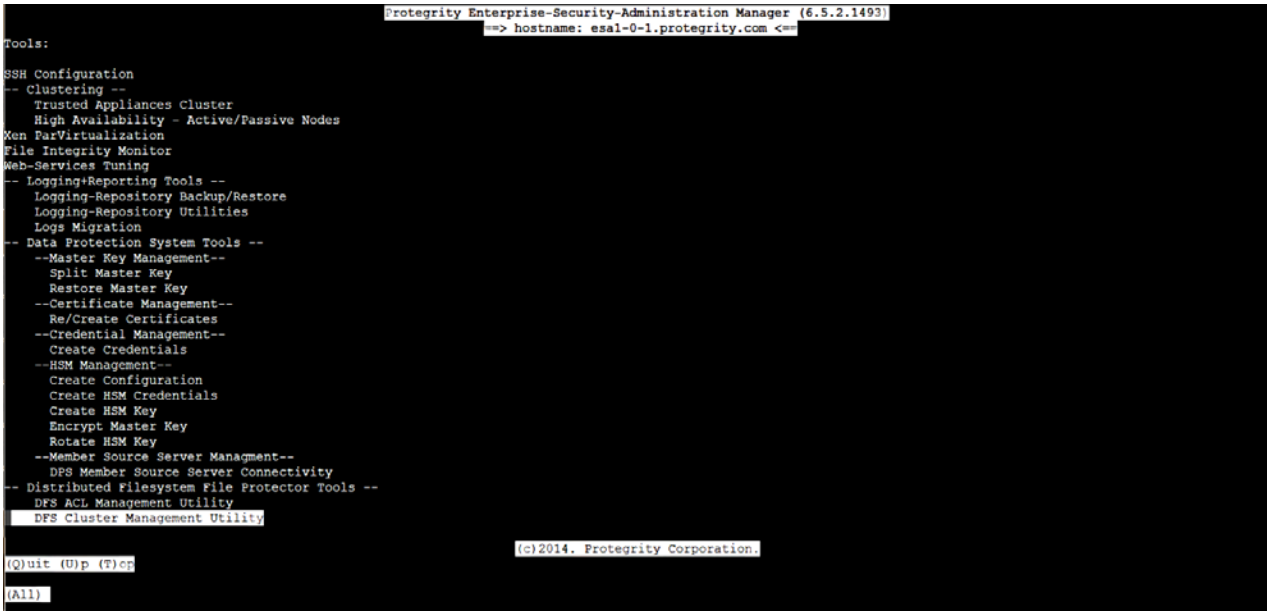
5.14 Using the DFS Cluster Management Utility (dfsdatastore)

The *dfsdatastore* utility enables you to manage the configuration of clusters on the ESA. The details of all options supported by this utility are described in this section.

5.14.1 Adding a Cluster for Protection

➢ **To add a cluster for protection using the *dfsdatastore* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Tools➢ DFS Cluster Management Utility**.

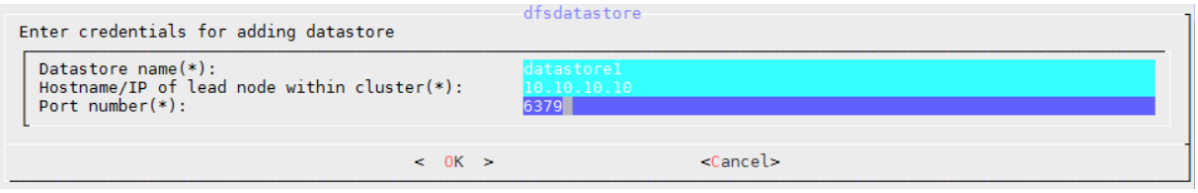


3. Press ENTER.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.
   The *dfsdatastore* UI appears.

6. Select the option *Add.*
7. Select **Next**.
8. Press ENTER.
   The *dfsdatastore* credentials screen appears.
9. Enter the following parameters:
   - Datastore name – The name for the datastore or cluster.

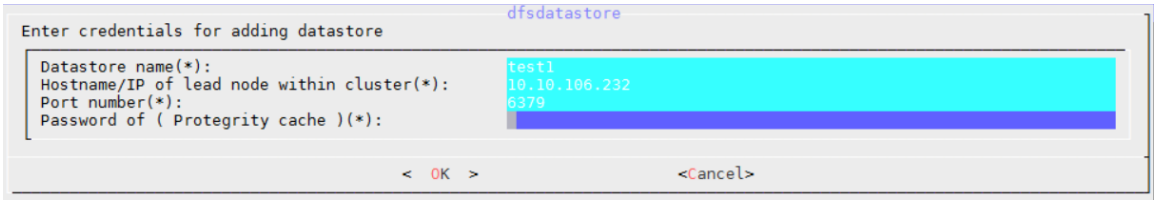
This name will be used for managing ACLs for the cluster.

- Hostname/IP of the Lead node within the cluster – The hostname or IP address of the Lead node of the cluster.
- Port number – The Protegrity Cache port which was specified in the *BDP.config* file during installation.



10. Select **OK**.
11. Press ENTER.
    The cluster with the specified parameters is added.
12. If the *DfsCacheRefresh* service is already running, then the datastore is added in an activated state.
    If the *DfsCacheRefresh* service is not running, then the datastore is added in an inactive state. The datastore can be activated by starting the *DfsCacheRefresh* service.

If you are using Big Data Protector with version lower than 6.6.3, then on the *dfsdatastore* credentials screen, a prompt for the Protegrity Cache password appears. You need to specify the Protegrity Cache password that was provided during the installation of the Big Data Protector.



➤ To start the DfsCacheRefresh Service:

1. Login to the ESA Web UI.
2. Navigate to **System➤ Services**.
3. Start the *DfsCacheRefresh* service.

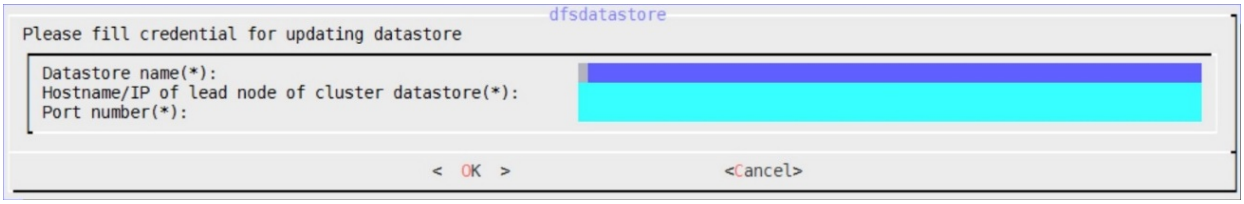
5.14.2 Updating a Cluster

Ensure that you utilize the **Update** option in the *dfsdatastore* UI to modify the parameters of an existing datastore only.

➤ To update a cluster using the *dfsdatastore* UI from the ESA CLI Manager:

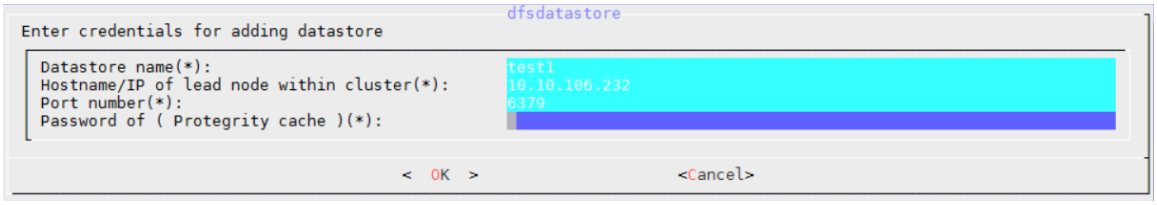
1. Login to the ESA CLI Manager.
2. Navigate to **Tools➤ DFS Cluster Management Utility**.
3. Press ENTER.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.
   The *dfsdatastore* UI appears.
6. Select the option *Update.*
7. Select **Next**.
8. Press ENTER.
   The *dfsdatastore* update screen appears.

9. Update the following parameters as required:
   - Hostname/IP of the Lead node within the cluster – The hostname or IP address of the Lead node of the cluster.
   - Port number – The Protegrity Cache port which was specified in the *BDP.config* file during installation.



10. Select **OK**.
11. Press ENTER.

If you are using Big Data Protector with version lower than 6.6.3, then on the *dfsdatastore* credentials screen, a prompt for the Protegrity Cache password appears. You need to specify the Protegrity Cache password that was provided during the installation of the Big Data Protector.



The cluster is modified with the required updates.

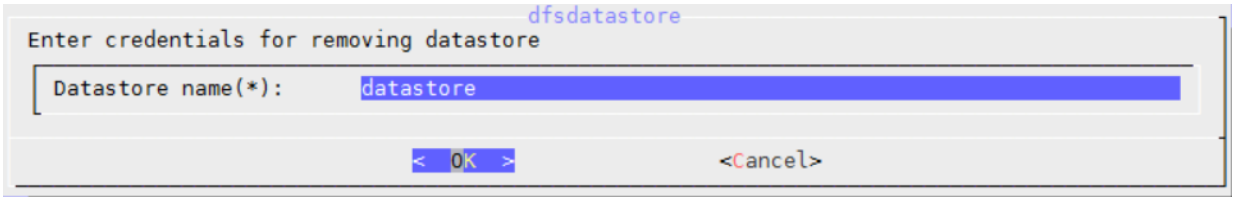
5.14.3 Removing a Cluster

Ensure that the *Cache Refresh Service* is running in the ESA Web UI before removing a cluster.

➤ **To remove a cluster using the *dfsdatastore* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Tools➤ DFS Cluster Management Utility**.
3. Press ENTER.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.
   The *dfsdatastore* UI appears.
6. Select the option *Remove.*
7. Select **Next**.
8. Press ENTER.
   The *dfsdatastore* remove screen appears.

9. Enter the following parameter:
   - Datastore name – The name for the datastore or cluster.
     This name will be used for managing ACLs for the cluster.

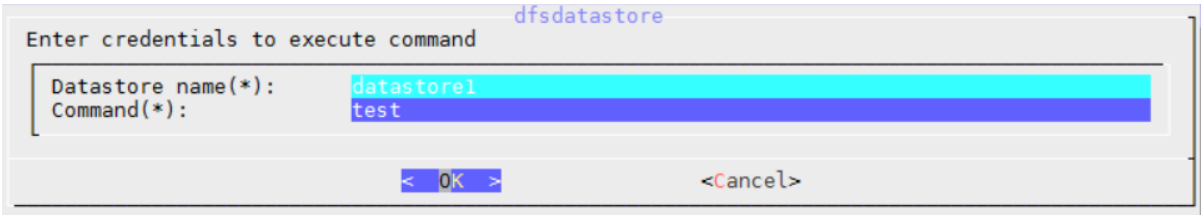


10. Select **OK**.
11. Press ENTER.
    The required cluster is removed.

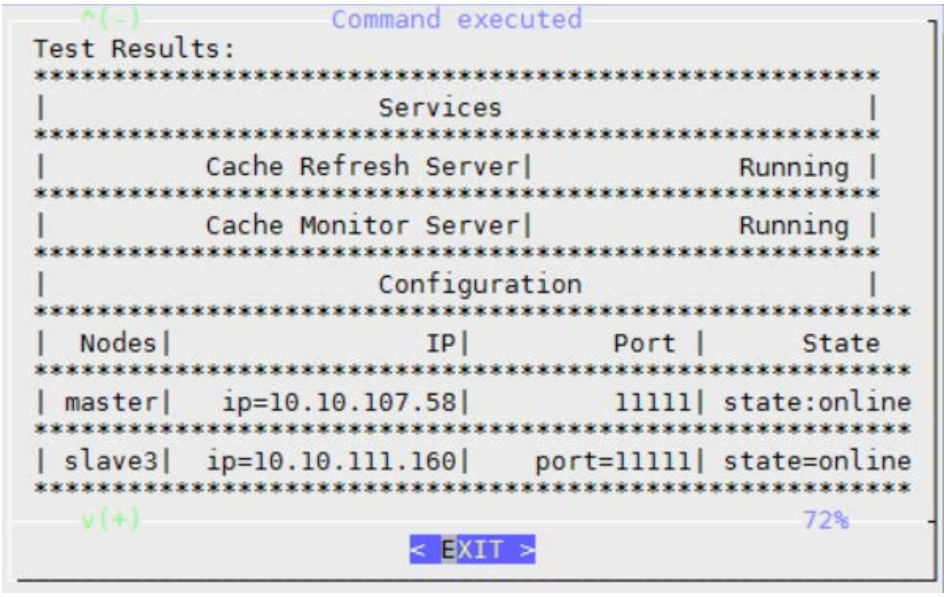
5.14.4 Monitoring a Cluster

➤ **To monitor a cluster using the *dfsdatastore* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Tools➤ DFS Cluster Management Utility**.
3. Press ENTER.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.
   The *dfsdatastore* UI appears.
6. Select the option *Execute Command.*
7. Select **Next**.
8. Press ENTER.
   The *dfsdatastore* execute command screen appears.
9. Enter the following parameters:
   - Datastore name - The name of the datastore or cluster.
     This name is used for managing ACLs for the cluster.
   - Command - The command to execute on the datastore. In this release, the only command supported is TEST. The command TEST is executed on the cluster, which is used to retrieve the statuses of the following servers:
     - Cache Refresh Server, running on the ESA
     - Cache Monitor Server, running on the Lead node of the cluster
     - Distributed Cache Server, running on the Lead and slave nodes of the cluster



10. Select **OK**.
11. Press ENTER.
    The *dfsdatastore* UI executes the TEST command on the cluster.

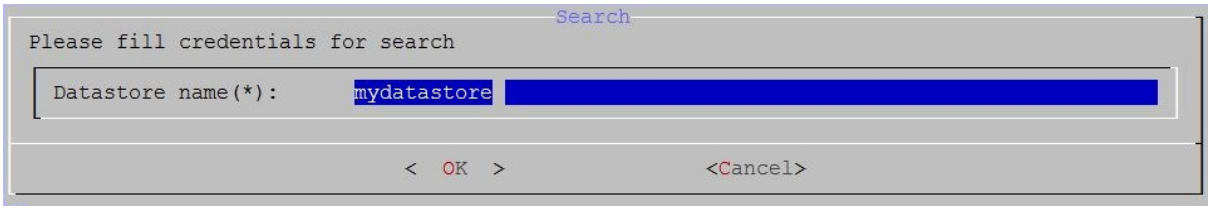


If you are using Big Data Protector with version lower than 6.6.3, then the Cluster Monitoring feature is not supported.

5.14.5 Searching a Cluster

➤ **To search for a cluster using the *dfsdatastore* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Tools➤ DFS Cluster Management Utility**.
3. Press ENTER.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.
   The *dfsdatastore* UI appears.
6. Select the option *Search.*
7. Select **Next**.
8. Press ENTER.
   The *dfsdatastore* search screen appears.
9. Enter the following parameter:
   - Datastore name – The name of the datastore or cluster.
     This name is used for managing ACLs for the cluster.



10. Select **OK**.
11. Press ENTER.
    The *dfsdatastore* UI searches for the required cluster.

5.14.6 Listing all Clusters

➤ **To list all clusters using the *dfsdatastore* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Tools➤ DFS Cluster Management Utility**.
3. Press ENTER.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.
   The *dfsdatastore* UI appears.
6. Select the option *List*.
7. Select **Next**.
8. Press ENTER.
   A list of all the clusters appears. Each cluster description contains one of the following cluster statuses:
   - 1: Cluster is in active state
   - 0: Cluster is in inactive state

5.15 Using the ACL Management Utility (dfsadmin)

The *dfsadmin* utility enables you to manage ACLs for cluster. Managing ACLs is a two-step process, which is creating or modifying ACL entries and then activating it. The protection of file or folder paths will not take effect until ACL entries are verified, confirmed and activated.

Ensure that an unstructured policy is created in the ESA, which is to be linked with the ACL.

The details of all options supported by this utility are described in this section.

5.15.1 Adding an ACL Entry for Protecting Directories in HDFS

It is recommended to not create ACLs for file paths.

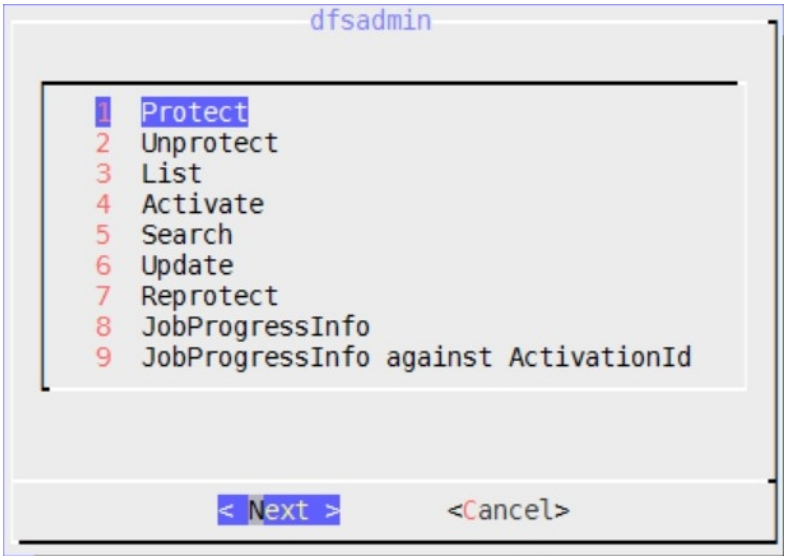
If the ACL for the directory, which is containing a file for which an ACL already exists, is being unprotected, then a decryption failure might occur, if there is a mismatch between the data elements used for the protection of the directory and the file contained in the directory.

➤ **To add an ACL entry for protecting files or folders using the *dfsadmin* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.

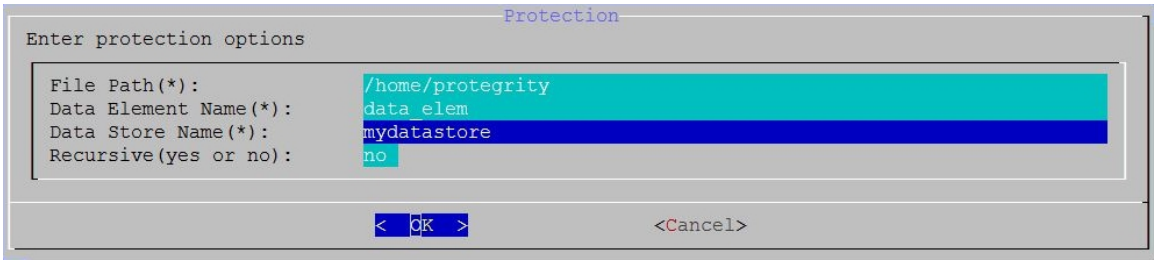
2. Navigate to **Tools➢ DFS ACL Management Utility**.

3. Press ENTER.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.
   The *dfsadmin* UI appears.



6. Select the option *Protect.*
7. Select **Next**.
8. Press ENTER.
   The *dfsadmin* protection screen appears.
9. Enter the following parameters:
   - File Path – The directory path to protect.
   - Data Element Name – The unstructured data element name to protect the HDFS directory path with.
   - Datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.

- Recursive (yes or no) – Select one of the following options:
  - Yes – Protect all files, sub-directories and their files, in the directory path.
  - No – Protect the files in the directory path.



10. Select **OK**.
11. Press ENTER.

    The ACL entry required for protecting the directory path is added to the *Inactive* list. The ACL entries can be activated by selecting the *Activate* option.

After the ACL entries are activated, the following actions occur, as required:

- If the recursive flag is not set, then all files inside the directory path are protected.
- If the recursive flag is set, then all the files, sub-directories and its files, in the directory path are protected.

If any MapReduce jobs or HDFS file shell commands are initiated on the ACL paths before the ACLs are activated, then the jobs or commands will fail. After the ACLs are activated, any new files that are ingested in the respecticve ACL directory path will get protected.

5.15.2 Updating an ACL Entry

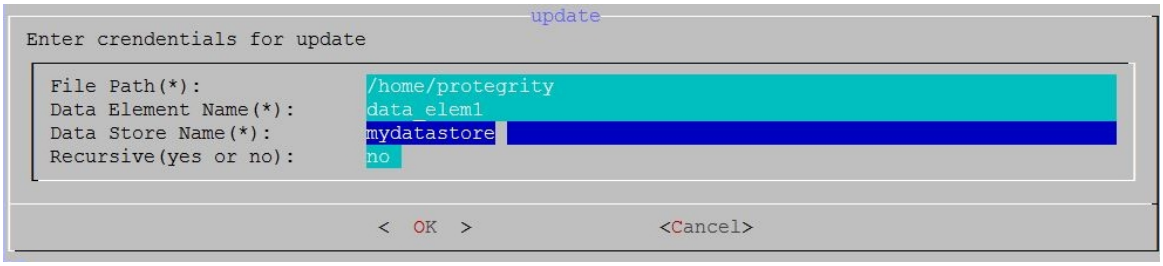
➤ **To update an ACL entry using the *dfsadmin* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Tools➤ DFS ACL Management Utility**.
3. Press ENTER.

   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.

   The *dfsadmin* UI appears.
6. Select the option *Update.*
7. Select **Next**.
8. Press ENTER.

   The *dfsadmin* update screen appears.
9. Update the following parameters as required:
   - File Path – The directory path to protect.
   - Datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.
   - Recursive (yes or no) – Select one of the following options:
     - Yes – Protect all files, sub-directories and their files, in the directory path.
     - No – Protect the files in the directory path.

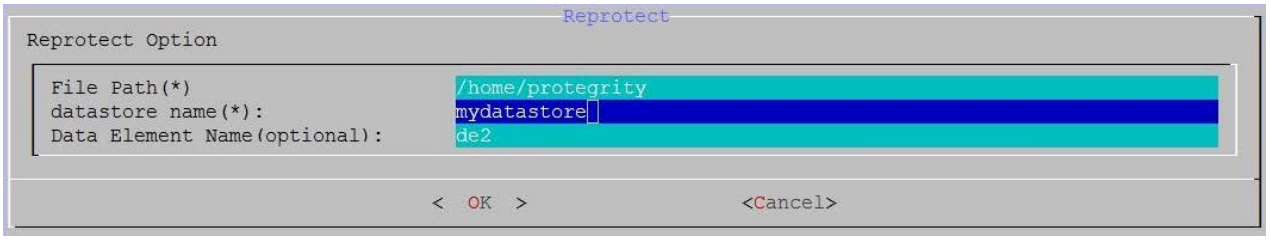


10. Select **OK**.
11. Press ENTER.
    The ACL entry is updated as required.

5.15.3 Reprotecting Files or Folders

➤ **To reprotect files or folders using the *dfsadmin* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Tools➤ DFS ACL Management Utility**.
3. Press ENTER.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.
   The *dfsadmin* UI appears.
6. Select the option *Reprotect.*
7. Select **Next**.
8. Press ENTER.
   The *dfsadmin* reprotection screen appears.
9. Enter the following parameters:
   - File Path – The directory path to protect.
   - datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.
   - Data Element Name – The data element name to protect the directory path with.
     If the user has rotated the data element key and needs to reprotect the data, then this field is optional.



10. Select **OK**.
11. Press ENTER.
    The files inside the ACL entry are reprotected.

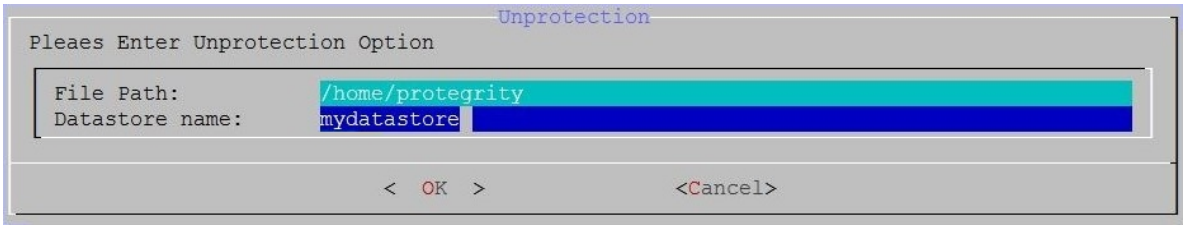
5.15.4 Deleting an ACL Entry to Unprotect Files or Directories

➤ **To delete an ACL entry to unprotect files or directories using the *dfsadmin* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Tools➤ DFS ACL Management Utility**.
3. Press ENTER.

The *root* password screen appears.

4. Enter the *root* password.
5. Press ENTER.
   The *dfsadmin* UI appears.
6. Select the option *Unprotect.*
7. Select **Next**.
8. Press ENTER.
   The *dfsadmin* unprotection screen appears.
9. Enter the following parameters as required:
   - File Path – The directory path which is protected.
   - Datastore Name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.



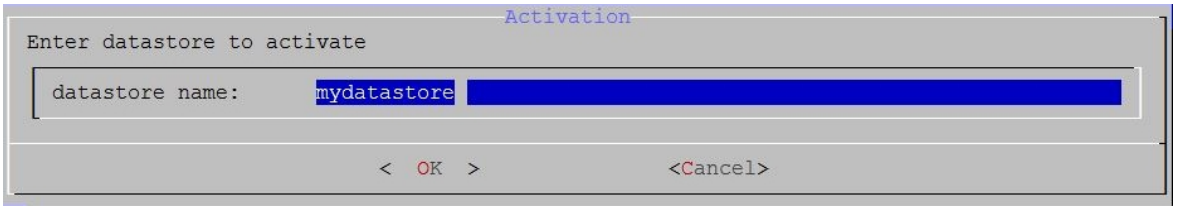
10. Select **OK**.
11. Press ENTER.
    The files inside the ACL entry are unprotected.

5.15.5 Activating Inactive ACL Entries

If you are using a Kerberos-enabled Hadoop cluster, then ensure that the user *ptyitusr* has a valid Kerberos ticket and write access permissions on the HDFS path for which the ACL is being created.

➢ **To activate inactive ACL entries using the *dfsadmin* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Tools➢ DFS ACL Management Utility**.
3. Press ENTER.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.
   The *dfsadmin* UI appears.
6. Select the option *Activate.*
7. Select **Next**.
8. Press ENTER.
   The *dfsadmin* activation screen appears.
9. Enter the following parameter as required:
   - datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.



10. Select **OK**.
11. Press ENTER.

    The inactive ACL entries in the Datastore are activated.

If an ACL entry for a directory containing files is activated (for Protect/Unprotect/Reprotect), then the ownerships and permissions for only the files contained in the directory are changed.

To avoid this issue, ensure that the user configured in the *PROTEGRITY_IT_USR* property in the *BDP.config* file during the Big Data Protector installation is added to the HDFS superuser group by running the following command on the Lead node:

```
usermod -a -G hdfs <User_configured_in_the_"PROTEGRITY_IT_USR" property>
```

If the protect or unprotect operation fails on the files or folders, which are a part of the ACL entry being activated and the message *ACL is locked* appears, then ensure that you monitor the *beuler.log* file for any exceptions and take the required corrective action.

➤ **To monitor the *beuler.log* file:**

1. Login to the Lead node with *root* permissions.
2. Switch the user to PROTEGRITY_IT_USR, as configured in the *BDP.config* file.
3. Navigate to the *<PROTEGRITY_DIR>/hdfsfp/ptyitusr* directory.
4. Monitor the *beuler.log* file for any exceptions.
5. If any exceptions appear in the *beuler.log* file, then resolve the exceptions as required.
6. Login to the Lead node and run the *beuler.sh* script.

   The following is a sample *beuler.sh* script command.

   ```
   sh beuler.sh -path <ACL_directory_path> -datastore <datastore_name> -activationid <activation_ID> -beulerjobid <beuler_job_ID>
   ```

   Alternatively, you can restart the *DfsCacheRefresh* service.

➤ **To restart the DfsCacheRefresh Service:**

1. Login to the ESA Web UI.
2. Navigate to **System➤ Services**.
3. Restart the *DfsCacheRefresh* service.

5.15.6 Viewing the ACL Activation Job Progress Information in the Interactive Mode

➤ **To view the ACL Activation Job Progress Information in the Interactive mode using the *dfsadmin* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Tools➤ DFS ACL Management Utility**.
3. Press ENTER.

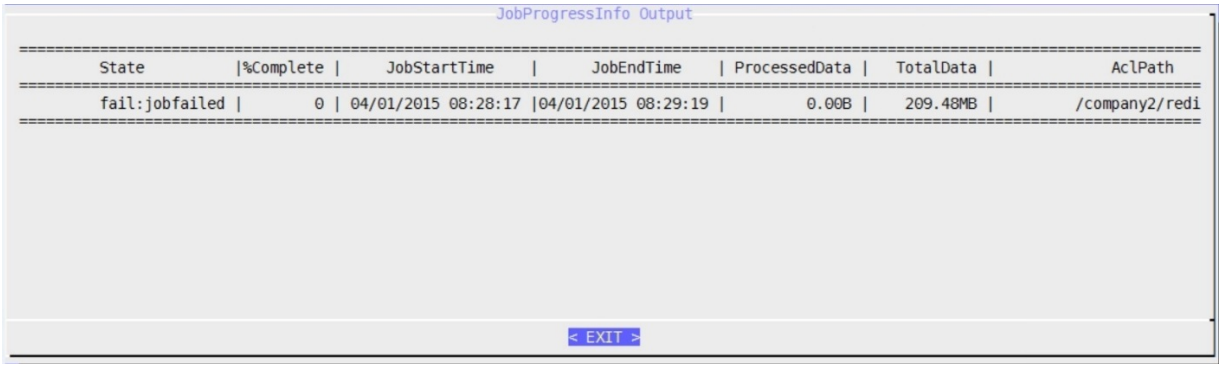
   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.

   The *dfsadmin* UI appears.
6. Select the option *JobProgressInfo.*
7. Select **Next**.
8. Press ENTER.

The *Activation ID* screen appears.

9. Enter the Activation ID.
10. Press ENTER.
    The filter search criteria screen appears.
11. If you need to specify the filtering criteria, then perform the following steps.
    c) Type *Y* or *y*.
    d) Select one of the following filtering criteria:
    - Start Time
    - Status
    - ACL Path
12. Select **Next**.
13. Press ENTER.
14. If you do not need to specify the search criteria, then type *N* or *n*.
    The *dfsadmin* job progress information screen appears listing all the jobs against the required Activation ID with the following information:

- State: One of the following states of the job:
  - Started
  - Failed
  - In progress
  - Completed
  - Yet to start
  - Failed as Path Does not Exist
- Percentage Complete: The percentage completion for the directory encryption
- Job Start Time: The time when the directory encryption started
- Job End Time: The time when the directory encryption ended
- Processed Data: The amount of data that is encrypted
- Total Data: The total directory size being encrypted
- ACL Path: The directory path being encrypted



5.15.7 Viewing the ACL Activation Job Progress Information in the Non Interactive Mode

➤ **To view the ACL Activation Job Progress Information in the Non Interactive mode using the *dfsadmin* UI from the ESA CLI Manager:**

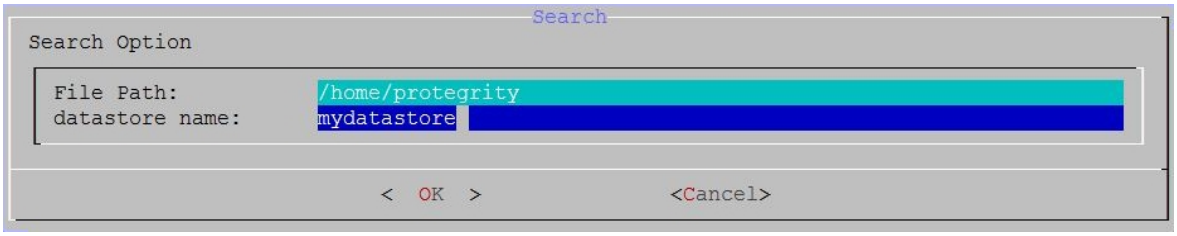
1. Login to the ESA CLI Manager.
2. Navigate to **Tools➤ DFS ACL Management Utility**.
3. Press ENTER.
   The *root* password screen appears.

4. Enter the *root* password.
5. Press ENTER.
   The *dfsadmin* UI appears.
6. Select the option *JobProgressInfo against ActivationId.*
   The *JobProgressInfo* Activation ID screen appears.
7. Enter the Activation ID.
8. Select **OK**.
9. Press ENTER.
   The *dfsadmin* job progress information screen for the required Activation ID appears.

5.15.8 Searching ACL Entries

➤ **To search for ACL entries using the *dfsadmin* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Tools➤ DFS ACL Management Utility**.
3. Press ENTER.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.
   The *dfsadmin* UI appears.
6. Select the option *Search.*
7. Select **Next**.
8. Press ENTER.
   The *dfsadmin* search screen appears.
9. Enter the following parameters as required:
   - File Path – The directory path to protect.
   - datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.



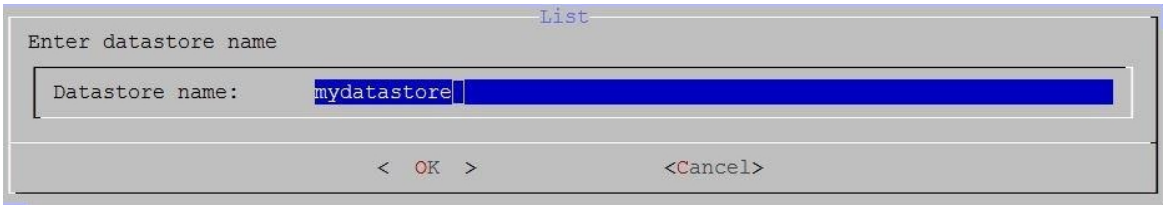
10. Select **OK**.
11. Press ENTER.
    The *dfsadmin* UI searches for the required ACL entry.

5.15.9 Listing all ACL Entries

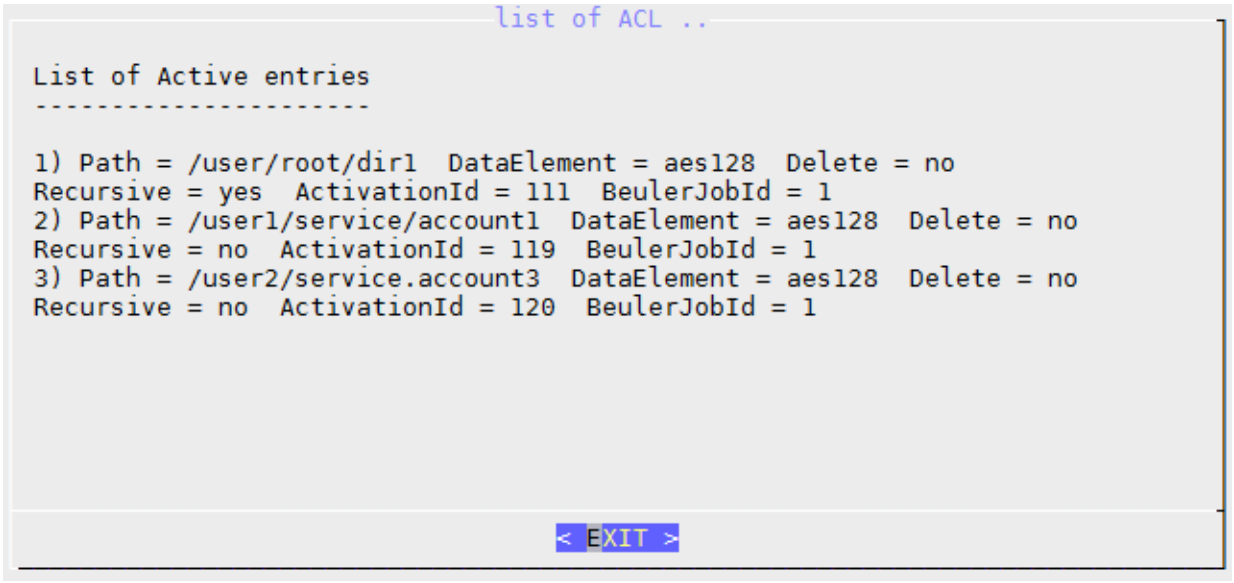
➤ **To list ACL entries using the *dfsadmin* UI from the ESA CLI Manager:**

1. Login to the ESA CLI Manager.
2. Navigate to **Tools➤ DFS ACL Management Utility**.
3. Press ENTER.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press ENTER.
   The *dfsadmin* UI appears.

6. Select the option *List*.
7. Select **OK**.
8. Press ENTER.
   The *dfsadmin* list screen appears.
9. Enter the following parameter as required:
   - Datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.



10. Select **OK**.
11. Press ENTER.
    A list of all the ACL entries appears.



If you are using Big Data Protector with version lower than 6.6.3, then the *List* option does not show the *Activation ID* and the *Beuler Job ID* for the respective ACLs.

5.16 HDFS Codec for Encryption and Decryption

A codec is an algorithm which provides compression and decompression. Hadoop provides a codec framework to compress blocks of data before storage. The codec compresses data while writing the blocks and decompresses data while reading the blocks.

A split-able codec is an algorithm which is applied after splitting a file, making it possible to recover original data from any part of the split.

The Protegrity HDFS codec is a split-able cryptographic codec. It uses encryption, such as AES 128, AES 256, DES and so on. It utilizes the infrastructure of the Protegrity Application Protector for applying cryptographic support. The protection is governed by the Policy deployed by the ESA, as defined by the Security Officer.

6 HBase

HBase is a database, which provides random read and write access to tables, consisting of rows and columns, in real-time. HBase is designed to run on commodity servers, to automatically scale as more servers are added, and is fault tolerant as data is divided across servers in the cluster. HBase tables are partitioned into multiple regions. Each region stores a range of rows in the table. Regions contain a datastore in memory and a persistent datastore (HFile). The Name node assigns multiple regions to a region server. The Name node manages the cluster and the region servers store portions of the HBase tables and perform the work on the data.

6.1 Overview of the HBase Protector

The Protegrity HBase protector extends the functionality of the data storage framework and provides transparent data protection and unprotection using coprocessors, which provide the functionality to run code directly on region servers. The Protegrity coprocessor for HBase runs on the region servers and protects the data stored in the servers. All clients which work with HBase are supported.

The data is transparently protected or unprotected, as required, utilizing the coprocessor framework.

6.2 HBase Protector Usage

The Protegrity HBase protector utilizes the *get*, *put*, and *scan* commands and calls the Protegrity coprocessor for the HBase protector. The Protegrity coprocessor for the HBase protector locates the metadata associated with the requested column qualifier and the current logged in user. If the data element is associated with the column qualifier and the current logged in user, then the HBase protector processes the data in a row based on the data elements defined by the security policy deployed in the Big Data Protector.

The Protegrity HBase coprocessor only supports *bytes* converted from the *string* data type.

If any other data type is directly converted to *bytes* and inserted in an HBase table, which is configured with the Protegrity HBase coprocessor, then data corruption might occur.

6.3 Adding Data Elements and Column Qualifier Mappings to a New Table

In an HBase table, every column family of a table stores metadata for that family, which contain the column qualifier and data element mappings.

Users need to add metadata to the column families for defining mappings between the data element and column qualifier, when a new HBase table is created.

The following command creates a new HBase table with one column family.

```
create 'table', { NAME => 'column_family_1', METADATA => {
'DATA_ELEMENT:credit_card'=>'CC_NUMBER','DATA_ELEMENT:name'=>'TOK_CUSTOMER_NAME'
} }
```

Parameters

table: Name of the table.

column_family_1: Name of the column family.

METADATA: Data associated with the column family.

DATA_ELEMENT: Contains the column qualifier name. In the example, the column qualifier names *credit_card* and *name*, correspond to data elements CC_NUMBER and TOK_CUSTOMER_NAME respectively.

6.4 Adding Data Elements and Column Qualifier Mappings to an Existing Table

Users can add data elements and column qualifiers to an existing HBase table. Users need to alter the table to add metadata to the column families for defining mappings between the data element and column qualifier.

The following command adds data elements and column qualifier mappings to a column in an existing HBase table.

```
alter 'table', { NAME => 'column_family_1', METADATA =>
{'DATA_ELEMENT:credit_card'=>'CC_NUMBER',
'DATA_ELEMENT:name'=>'TOK_CUSTOMER_NAME' } }
```

Parameters

table: Name of the table.

column_family_1: Name of the column family.

METADATA: Data associated with the column family.

DATA_ELEMENT: Contains the column qualifier name. In the example, the column qualifier names *credit_card* and *name*, correspond to data elements CC_NUMBER and TOK_CUSTOMER_NAME respectively.

6.5 Inserting Protected Data into a Protected Table

Users can ingest protected data into a protected table in HBase using the BYPASS_COPROCESSOR flag. If the BYPASS_COPROCESSOR flag is set while inserting data in the HBase table, then the Protegrity coprocessor for HBase is bypassed.

The following command bypasses the Protegrity coprocessor for HBase and ingests protected data into an HBase table.

```
put 'table', 'row_2', 'column_family:credit_card', '3603144224586181', {
ATTRIBUTES => {'BYPASS_COPROCESSOR'=>'1'}}
```

Parameters

table: Name of the table.

column_family: Name of the column family and the protected data to be inserted in the column.

METADATA: Data associated with the column family.

ATTRIBUTES: Additional parameters to consider when ingesting the protected data. In the example, the flag to bypass the Protegrity coprocessor for HBase is set.

6.6 Retrieving Protected Data from a Table

If users need to retrieve protected data from an HBase table, then they need to set the BYPASS_COPROCESSOR flag to retrieve the data. This is necessary to retain the protected data as is since HBase performs protects and unprotects the data transparently.

The following command bypasses the Protegrity coprocessor for HBase and retrieves protected data from an HBase table.

```
scan 'table', { ATTRIBUTES => {'BYPASS_COPROCESSOR'=>'1'}}
```

Parameters

table: Name of the table.

ATTRIBUTES: Additional parameters to consider when ingesting the protected data. In the example, the flag to bypass the Protegrity coprocessor for HBase is set.

6.7 Protecting Existing Data

Users should define the mappings between the data elements and column qualifiers in the respective column families, which are used to by the coprocessor to protect or unprotect the data.

The following command protects the existing data in an HBase table by setting the MIGRATION flag. Data from the table is read, protected, and inserted back into the table.

```
scan 'table', { ATTRIBUTES => {'MIGRATION'=>'1'}}
```

Parameters

table: Name of the table.

ATTRIBUTES: Additional parameters to consider when ingesting the protected data. In the example, the Migration flag is set to protect the existing data in the HBase table.

6.8 HBase Commands

Hadoop provides shell commands to ingest, extract, and display the data in an HBase table.

The section describes the commands supported by HBase.

6.8.1 put

This command ingests the data provided by the user in protected form, using the configured data elements, into the required row and column of an HBase table. You can use this command to ingest data into all the columns for the required row of the HBase table.

```
put '<table_name>','<row_number>', 'column_family:<column_name>', '<data>'
```

Parameters

table_name: Name of the table.

row_number: Number of the row in the HBase table.

column_family: Name of the column family and the protected data to be inserted in the column.

6.8.2 get

This command displays the protected data from the required row and column of an HBase table in cleartext form. You can use this command to display the data contained in all the columns of the required row of the HBase table.

```
get '<table_name>','<row_number>', 'column_family:<column_name>'
```

Parameters

table_name: Name of the table.

row_number: Number of the row in the HBase table.

column_family: Name of the column family.

Ensure that the logged in user has the permissions to view the protected data in cleartext form. If the user does not have the permissions to view the protected data, then only the protected data appears.

6.8.3 scan

This command displays the data from the HBase table in protected or unprotected form.

View the protected data using the following command.

```
scan '<table_name>', { ATTRIBUTES => {'BYPASS_COPROCESSOR'=>'1'}}
```

View the unprotected data using the following command.

```
scan '<table_name>'
```

Parameters

table_name: Name of the table.

ATTRIBUTES: Additional parameters to consider when displaying the protected or unprotected data.

Ensure that the logged in user has the permissions to unprotect the protected data. If the user does not have the permissions to unprotect the protected data, then only the protected data appears.

6.9 Ingesting Files Securely

To ingest data into HBase securely, use the *put* command.

For more information, refer to section *6.8.1 put*.

6.10 Extracting Files Securely

To extract data from HBase securely, use the *get* command.

For more information, refer to section *6.8.2 get*.

6.11 Sample Use Cases

For information about the HBase protector sample use cases, refer to section *12.8 Protecting Data using HBase*.

7 Impala

Impala is an MPP SQL query engine for querying the data stored in a cluster. It provides the flexibility of the SQL format and is capable of running the queries on HDFS in HBase.

This section provides information about the Impala protector, the UDFs provided, and the commands for protecting and unprotecting data in an Impala table.

7.1 Overview of the Impala Protector

Impala is an MPP SQL query engine for querying the data stored in a cluster. The Protegrity Impala protector extends the functionality of the Impala query engine and provides UDFs which protect or unprotect the data as it is stored or retrieved.

7.2 Impala Protector Usage

The Protegrity Impala protector provides UDFs for protecting data using encryption or tokenization, and unprotecting data by using decryption or detokenization.

Ensure that the */user/impala* path exists in HDFS with the Impala supergroup permissions.

You can verify this by the following command:

```
# hadoop fs -ls /user
```

➤ **To create the */user/impala* path in Impala with Supergroup permissions:**

If the */user/impala* path does not exist or does not have supergroup permissions, then perform the following steps.

1. Create the */user/impala* directory in HDFS using the following command.
   ```
   # sudo -u hdfs hadoop -mkdir /user/impala
   ```
2. Assign Impala supergroup permissions to the */user/impala* path using the following command.
   ```
   # sudo -u hdfs hadoop -chown -R impala:supergroup /user/impala
   ```

7.3 Impala UDFs

This section describes all Impala UDFs that are available for protection and unprotection in Big Data Protector to build secure Big Data applications.

7.3.1 pty_GetVersion()

This UDF returns the PEP version number.

ptyGetVersion()

Parameters

None

Result

This UDF returns the current version of the PEP.

Example

```
select pty_GetVersion ();
```

7.3.2 pty_WhoAmI()

This UDF returns the logged in user name.

ptyWhoAmI()

Parameters

None

Result

`Text`: Returns the logged in user name

Example

```
select pty_WhoAmI();
```

7.3.3 pty_GetCurrentKeyId()

This UDF returns the current active key identification number of the encryption type data element.

pty_GetCurrentKeyId(dataElement string)

Parameters

`dataElement`: Variable specifies the protection method

Result

`integer`**:** Returns the current key identification number

Example

```
select pty_GetCurrentKeyId('enc_3des_kid');
```

7.3.4 pty_GetKeyId()

This UDF returns the key ID used for each row in a table.

pty_GetKeyId(dataElement string, col string)

Parameters

`dataElement`: Variable specifies the protection method
`col`: String array of the data in table

Result

`integer`: Returns the key indentification number for the row

Example

```
select pty_GetKeyId('enc_3des_kid',column_name) from table_name;
```

7.3.5 pty_StringEnc()

This UDF returns the encrypted value for a column containing *String* format data.

pty_StringEnc(data string, dataElement string)

Parameters

data: Column name of the data to encrypt in the table
dataElement: Variable specifying the protection method

Result

string: Returns a string value

Example

```
select pty_StringEnc(column_name,'enc_3des') from table_name;
```

7.3.6 pty_StringDec()

This UDF returns the decrypted value for a column containing *String* format data.

pty_StringDec(data string, dataElement string)

Parameters

data: Column name of the data to decode in the table
dataElement: Variable specifying the unprotection method

Result

string: Returns a string value

Example

```
select pty_StringDec(column_name,'enc_3des') from table_name;
```

7.3.7 pty_StringIns()

This UDF returns the tokenized value for a column containing *String* format data.

pty_StringIns(data string, dataElement string)

Parameters

data: Column name of the data to tokenize in the table
dataElement: Variable specifying the protection method

Result

string: Returns the tokenized string value

Example

```
select pty_StringIns(column_name, 'TOK_NAME') from table_name;
```

7.3.8 pty_StringSel()

This UDF returns the detokenized value for a column containing *String* format data.

pty_StringSel(data string, dataElement string)

Parameters

data: Column name of the data to detokenize in the table
dataElement: Variable specifing the unprotection method

Result

string: Returns the detokenized string value

Example

```
select pty_StringSel(column_name, 'TOK_NAME') from table_name;
```

7.3.9 pty_UnicodeStringIns()

This UDF returns the tokenized value for a column containing *String* (Unicode) format data.

pty_UnicodeStringIns(data string, dataElement string)

Parameters

`data:` Column name of the *string* (Unicode) format data to tokenize in the table

`dataElement:` Name of data element to protect *string* (Unicode) value

This UDF should be used only if you need to tokenize Unicode data in Impala, and migrate the tokenized data from Impala to a Teradata database and detokenize the data using the Protegrity Database Protector.

Ensure that you use this UDF with a Unicode tokenization data element only.

For more information about migrating tokenized Unicode data to a Teradata database, refer to section *15 Appendix: Migrating Tokenized Unicode Data from and to a Teradata Database*.

Result

This UDF returns protected *string* value.

Example

```
select pty_UnicodeStringIns(val, 'Token_unicode') from temp_table;
```

7.3.10 pty_UnicodeStringSel()

This UDF unprotects the existing protected *String* value.

pty_UnicodeStringSel(data string, dataElement string)

Parameters

`data:` Column name of the *string* format data to detokenize in the table

`varchar dataElement:` Name of data element to unprotect *string* value

This UDF should be used only if you need to tokenize Unicode data in Teradata using the Protegrity Database Protector, and migrate the tokenized data from a Teradata database to Impala and detokenize the data using the Protegrity Big Data Protector for Impala.

Ensure that you use this UDF with a Unicode tokenization data element only.

For more information about migrating tokenized Unicode data from a Teradata database, refer to section *15 Appendix: Migrating Tokenized Unicode Data from and to a Teradata Database*.

Result

This UDF returns detokenized *string* (Unicode) value.

Example

```
select pty_UnicodeStringSel(val, 'Token_unicode') from temp_table;
```

7.3.11 pty_IntegerEnc()

This UDF returns the encrypted value for a column containing *Integer* format data.

pty_IntegerEnc(data integer, dataElement string)

Parameters

`data`: Column name of the data to encrypt in the table
`dataElement`: Variable specifying the protection method

Result

`string`**:** Returns a string value

Example

```
select pty_IntegerEnc(column_name,'enc_3des') from table_name;
```

7.3.12 pty_IntegerDec()

This UDF returns the decrypted value for a column containing *Integer* format data.

pty_IntegerDec(data string, dataElement string)

Parameters

`data`: Column name of the data to decode in the table
`dataElement`: Variable specifying the unprotection method

Result

`integer`**:** Returns an integer value

Example

```
select pty_IntegerDec(column_name,'enc_3des') from table_name;
```

7.3.13 pty_IntegerIns()

This UDF returns the tokenized value for a column containing *Integer* format data.

pty_IntegerIns(data integer, dataElement string)

Parameters

`data`: Column name of the data to tokenize in the table
`dataElement`: Variable specifying the protection method

Result

`integer`: Returns the tokenized integer value

Example

```
select pty_IntegerIns(column_name,'integer_de') from table_name;
```

7.3.14 pty_IntegerSel()

This UDF returns the detokenized value for a column containing *Integer* format data.

pty_IntegerSel(data integer, dataElement string)

Parameters

`data`: Column name of the data to detokenize in the table
`dataElement`: Variable specifing the unprotection method

Result

`integer`: Returns the detokenized integer value

Example

```
select pty_IntegerSel(column_name,'integer_de') from table_name;
```

7.3.15 pty_FloatEnc()

This UDF returns the encrypted value for a column containing *Float* format data.

pty_FloatEnc(data float, dataElement string)

Parameters

`data`: Column name of the data to encrypt in the table
`dataElement`: Variable specifying the protection method

Result

`string`**:** Returns a string value

Example

```
select pty_FloatEnc(column_name,'enc_3des') from table_name;
```

7.3.16 pty_FloatDec()

This UDF returns the decrypted value for a column containing *Float* format data.

pty_FloatDec(data string, dataElement string)

Parameters

`data`: Column name of the data to decode in the table
`dataElement`: Variable specifying the unprotection method

Result

`float`**:** Returns a float value

Example

```
select pty_FloatDec(column_name,'enc_3des') from table_name;
```

7.3.17 pty_FloatIns()

This UDF returns the tokenized value for a column containing *Float* format data.

pty_FloatIns(data float, dataElement string)

Parameters

`data`: Column name of the data to tokenize in the table
`dataElement`: Variable specifying the protection method

Result

`float`: Returns the tokenized float value

Example

```
select pty_FloatIns(cast(12.3 as float), 'no_enc');
```

Ensure that you use the data element with the *No Encryption* method only. Using any other data element would return an error mentioning that the operation is not supported for that data type.

If you need to tokenize the Float column, then load the Float column into a String column and use the pty_StringIns UDF to tokenize the column.

For more information about the pty_StringIns UDF, refer to section *7.3.7 pty_StringIns()*.

7.3.18 pty_FloatSel()

This UDF returns the detokenized value for a column containing *Float* format data.

pty_FloatSel(data float, dataElement string)

Parameters

`data`: Column name of the data to detokenize in the table
`dataElement`: Variable specifing the unprotection method

Result

`float`: Returns the detokenized float value

Example

```
select pty_FloatSel(tokenized_value, 'no_enc');
```

Ensure that you use the data element with the *No Encryption* method only. Using any other data element would return an error mentioning that the operation is not supported for that data type.

7.3.19 pty_DoubleEnc()

This UDF returns the encrypted value for a column containing *Double* format data.

pty_DoubleEnc(data double, dataElement string)

Parameters

`data`: Integer data column to encrypt in the table
`dataElement`: Variable specifying the protection method

Result

`string`**:** Returns a string

Example

```
select pty_DoubleEnc(column_name,'enc_3des') from table_name;
```

7.3.20 pty_DoubleDec()

This UDF returns the decrypted value for a column containing *Double* format data.

Pty_DoubleDec(data string, dataElement string)

Parameters

`data`: Column name of the data to decode in the table
`dataElement`: Variable specifying the unprotection method

Result

`double`**:** Returns a *double* value

Example

```
select pty_DoubleDec(column_name,'enc_3des') from table_name;
```

7.3.21 pty_DoubleIns()

This UDF returns the tokenized value for a column containing *Double* format data.

pty_DoubleIns(data double, dataElement string)

Parameters

`data`: Column name of the data to tokenize in the table
`dataElement`: Variable specifying the protection method

Result

`double`: Returns a *double* value

Example

```
select pty_DoubleIns(cast(1.2 as double), 'no_enc');
```

Ensure that you use the data element with the *No Encryption* method only. Using any other data element would return an error mentioning that the operation is not supported for that data type.

If you need to tokenize the *Double* column, then load the *Double* column into a String column and use the pty_StringIns UDF to tokenize the column.

For more information about the pty_StringIns UDF, refer to section *7.3.7 pty_StringIns()*.

7.3.22 pty_DoubleSel()

This UDF returns the detokenized value for a column containing *Double* format data.

pty_DoubleSel(data double, dataElement string)

Parameters

`data`: Column name of the data to detokenize in the table
`dataElement`: Variable specifing the unprotection method

Result

`double`: Returns the detokenized *double* value

Example

```
select pty_DoubleSel(tokenized_value, 'no_enc');
```

Ensure that you use the data element with the *No Encryption* method only. Using any other data element would return an error mentioning that the operation is not supported for that data type.

7.4 Inserting Data from a File into a Table

To insert data from a file into an Impala table, ensure that the required user permissions for the directory path in HDFS are assigned for the Impala table.

➤ **To prepare the environment for the *basic_sample.csv* file:**

1. Assign permissions to the path where data from the *basic_sample.csv* file needs to be copied using the following command:
   sudo -u hdfs hadoop fs -chown root:root /tmp/basic_sample/sample/
2. Copy the data from the *basic_sample.csv* file into HDFS using the following command:
   hdfs dfs -put basic_sample.csv /tmp/basic_sample/sample/
3. Verify the presence of the *basic_sample.csv* file in the HDFS path using the following command:
   hdfs dfs -ls /tmp/basic_sample/sample/
4. Assign permissions for Impala to the path where the *basic_sample.csv* file is located using the following command:
   sudo -u hdfs hadoop fs -chown impala:supergroup /path/

➤ **To populate the table *sample_table* from the *basic_sample_data.csv* file:**

The following commands populate the table *basic_sample* with the data from the *basic_sample_data.csv* file.

```
create table sample_table(colname1 colname1_format, colname2 colname2_format,
colname3 colname3_format)

row format delimited fields terminated by ',';

LOAD DATA INPATH '/tmp/basic_sample/sample/' INTO TABLE sample_table;
```

Parameters

sample_table: Name of the Impala table created to load the data from the input CSV file from the required path.

colname1, colname2, colname3: Name of the columns.

colname1_format, colname2_format, colname3_format: The data types contained in the respective columns. The data types can only be of types STRING, INT, DOUBLE, or FLOAT.

ATTRIBUTES: Additional parameters to consider when ingesting the data.

In the example, the row format is delimited using the ‘,’ character as the row format in the input file is comma separated. If the input file is tab separated, then the the row format is delimited using '\t'.

7.5 Protecting Existing Data

To protect existing data, users should define the mappings between the columns and their respective data elements in the data security policy.

The following commands ingest cleartext data from the *basic_sample* table to the *basic_sample_protected* table in protected form using Impala UDFs.

```
create table basic_sample_protected (colname1 colname1_format, colname2
colname2_format, colname3 colname3_format)

insert into basic_sample_protected(colname1, colname2, colname3) select
ID,pty_stringins(colname1, dataElement1),pty_stringins(colname2,
dataElement2),pty_stringins(colname3, dataElement3) from basic_sample;
```

Parameters

basic_sample_protected: Table to store protected data.

colname1, colname2, colname3: Name of the columns.

dataElement1, dataElement2, dataElement3: The data elements corresponding to the columns.

basic_sample: Table containing the original data in cleartext form.

7.6 Unprotecting Protected Data

To unprotect protected data, you need to specify the name of the table which contains the protected data, the table which would store the unprotected data, and the columns and their respective data elements.

Ensure that the user performing the task has permissions to unprotect the data as required in the data security policy.

The following commands unprotect the protected data in a table and stores the data in cleartext form in to a different table, if the user has the required permissions.

```
create table table_unprotected (colname1 colname1_format, colname2
colname2_format, colname3 colname3_format)

insert into table_unprotected (colname1, colname2, colname3) select
ID,pty_stringsel(colname1, dataElement1),pty_stringsel(colname2,
dataElement2),pty_stringsel(colname3, dataElement3) from table_protected;
```

Parameters

table_unprotected: Table to store unprotected data.

colname1, colname2, colname3: Name of the columns.

dataElement1, dataElement2, dataElement3: The data elements corresponding to the columns.

table_protected: Table containing protected data.

7.7 Retrieving Data from a Table

To retrieve data from a table, the user needs to have access to the table.

The following command displays the data contained in the table.

```
select * from table;
```

Parameters

table: Name of the table.

7.8 Sample Use Cases

For information about the Impala protector sample use cases, refer to section *11.9 Protecting Data using Impala*.

8 HAWQ

HAWQ is an MPP SQL processing engine for querying the data stored in a Hadoop cluster. It breaks complex queries into smaller tasks and distributes their execution to the query processing units.

HAWQ is an MPP database, which uses HDFS to store data. It has the following components:

- HAWQ Master Server: Enables users to interact with HAWQ using client programs, such as PSQL or APIs, such as JDBC or ODBC.
  The HAWQ Master Server performs the following functions:
  - Authenticates client connections
  - Processes incoming SQL commands
  - Distributes workload among HAWQ segments
  - Coordinates the results returned by each segment
  - Presents the final results to the client application
- Name Node: Enables client applications to locate a file.
- HAWQ Segments: Are the units which process the individual data modules simultaneously
- HAWQ Storage: Is HDFS, which stores all the table data
- Interconnect Switch: Is the networking layer of HAWQ, which handles the communication between the segments

This section provides information about the HAWQ protector, the UDFs provided, and the commands for protecting and unprotecting data in a HAWQ table.

8.1 Overview of the HAWQ Protector

The Protegrity HAWQ protector extends the functionality of the HAWQ processing engine and provides UDFs which protect or unprotect the data as it is stored or retrieved.

8.2 HAWQ Protector Usage

The Protegrity HAWQ protector provides UDFs for protecting data using encryption or tokenization, and unprotecting data by using decryption or detokenization. Ensure that the format of the data is either *Varchar*, *Integer*, *Date*, or *Real*.

Ensure that the HAWQ is configured after the Big Data Protector is installed.

For more information about configuring HAWQ, refer to section *3.1.11 Configuring HAWQ.*

8.3 HAWQ UDFs

This section describes all HAWQ UDFs that are available for protection and unprotection in Big Data Protector to build secure Big Data applications.

8.3.1 pty_GetVersion()

This UDF returns the PEP version number.

Pty_GetVersion()

Parameters
None

Returns
This UDF returns the current PEP server version

Example

```
select pty_GetVersion();
```

8.3.2 pty_WhoAmI()

This UDF returns the logged in user name.

Pty_WhoAmI()

Parameters

None

Returns

This UDF returns the current logged in user name

Example

```
select pty_WhoAmI();
```

8.3.3 pty_GetCurrentKeyId()

This UDF returns the current active key identification number of the encryption type data element.

pty_GetCurrentKeyId (dataElement varchar)

Parameters

`dataElement`: Variable specifies the protection method

Returns

This UDF returns the current key identification number of the encryption type data element, which is passed as the parameter.

Example

```
select pty_GetCurrentKeyId('enc_de');
```

8.3.4 pty_GetKeyId()

This UDF returns the key ID for the encryption data element, used for protecting each row in a table.

pty_GetKeyId(dataElement string, col byte[])

Parameters

`dataElement`: Variable specifies the protection method

`col`: Byte array of the column in the table

Returns

This UDF returns the key ID for the encryption data element, used for protecting each row in the table

Example

```
select pty_GetKeyId('enc_de',table_name.c) from table_name;
```

8.3.5 pty_VarcharEnc()

This UDF returns the encrypted value for a column containing *varchar* format data.

pty_VarcharEnc(col varchar, dataElement varchar)

Parameters

col: Column name of the data to encrypt in the table
dataElement: Variable specifying the protection method

Returns

This UDF returns the encrypted value as a byte array

Example

```
select pty_VarcharEnc(column_name,'enc_de') from table_name;
```

8.3.6 pty_VarcharDec()

This UDF returns the decrypted value for a column containing *varchar* format protected data.

pty_VarcharDec(col byte[], dataElement varchar)

Parameters

col: Column name of the data to decrypt in the table
dataElement: Variable specifying the unprotection method

Returns

This UDF returns the decrypted value

Example

```
select pty_VarcharDec(column_name,'enc_de') from table_name;
```

8.3.7 pty_VarcharHash()

This UDF returns the hashed value for a column containing *varchar* format data.

pty_VarcharHash(col varchar, dataElement varchar)

Parameters

col: Column name of the data to hash in the table
dataElement: Variable specifying the protection method

Returns

The protected value as byte array

Example

```
select pty_VarcharHash(column_name,'hash_de') from table_name;
```

8.3.8 pty_VarcharIns()

This UDF returns the tokenized value for a column containing *varchar* format data.

pty_VarcharIns(col varchar, dataElement varchar)

Parameters

col: Column name of the data to tokenize in the table
dataElement: Variable specifying the protection method

Returns

This UDF returns the tokenized value as byte array

Example

```
select pty_VarcharIns(column_name,'alpha_num_tk_de') from table_name;
```

8.3.9 pty_VarcharSel()

This UDF returns the detokenized value for a column containing *varchar* format tokenized data.

pty_VarcharSel(col varchar, dataElement varchar)

Parameters

`col`: Column name of the data to detokenize in the table
`dataElement`: Variable specifying the unprotection method

Returns

This UDF returns the detokenized value

Example

```
select pty_VarcharSel(column_name,'alpha_num_tk_de') from table_name;
```

8.3.10 pty_UnicodeVarcharIns()

This UDF protects *varchar* (Unicode) values.

pty_UnicodeVarcharIns(col varchar, dataElement varchar)

Parameters

`col`: Column name of the *varchar* (Unicode) data to protect
`dataElement`: Name of data element to protect *varchar* (Unicode) data.

This UDF should be used only if you need to tokenize Unicode data in HAWQ, and migrate the tokenized data from HAWQ to a Teradata database and detokenize the data using the Protegrity Database Protector.

Ensure that you use this UDF with a Unicode tokenization data element only.

For more information about migrating tokenized Unicode data to a Teradata database, refer to section *15 Appendix: Migrating Tokenized Unicode Data from and to a Teradata Database*.

Result

This UDF returns protected *varchar* value.

Example

```
select pty_UnicodeVarcharIns(column_name, 'Token_unicode') from temp_table;
```

8.3.11 pty_UnicodeVarcharSel()

This UDF unprotects *varchar* values.

pty_unicodevarcharsel(col varchar, dataElement varchar)

Parameters

`varchar input`: Column name of the *varchar* data to unprotect
`varchar dataElement`: Name of data element to unprotect *varchar* data

This UDF should be used only if you need to tokenize Unicode data in Teradata using the Protegrity Database Protector, and migrate the tokenized data from a Teradata database to HAWQ and detokenize the data using the Protegrity Big Data Protector for HAWQ.

Ensure that you use this UDF with a Unicode tokenization data element only.

For more information about migrating tokenized Unicode data to a Teradata database, refer to section *15 Appendix: Migrating Tokenized Unicode Data from and to a Teradata Database*.

Result

This UDF returns unprotected *varchar* (Unicode) value.

Example

```
select pty_unicodevarcharsel(column_name, 'Token_unicode') from temp_table;
```

8.3.12 pty_IntegerEnc()

This UDF returns the encrypted value for a column containing *integer* format data.

pty_IntegerEnc(col integer, dataElement varchar)

Parameters

`col`: Column name of the data to encrypt in the table
`dataElement`: Variable specifying the protection method

Returns

This UDF returns the encrypted value as byte array

Example

```
select pty_IntegerEnc(column_name,'enc_de') from table_name;
```

8.3.13 pty_IntegerDec()

This UDF returns the decrypted value for a column containing encrypted data as byte array format.

pty_IntegerEnc(col byte[], dataElement varchar)

Parameters

`col`: Column name of the data to decrypt in the table
`dataElement`: Variable specifying the unprotection method

Returns

This UDF returns the decrypted value

Example

```
select pty_IntegerDec(column_name,'enc_de') from table_name;
```

8.3.14 pty_IntegerHash()

This UDF returns the hashed value for a column, containing integer format data, as a byte array.

pty_IntegerHash(col integer, dataElement varchar)

Parameters

`col`: Column name of the data to hash in the table
`dataElement`: Variable specifying the protection method

Returns

This UDF returns the hash value as byte array

Example

```
select pty_IntegerHash(column_name,'hash_de') from table_name;
```

8.3.15 pty_IntegerIns()

This UDF returns the tokenized value for a column containing *integer* format data.

pty_IntegerIns(col integer, dataElement varchar)

Parameters

`col`: Column name of the data to tokenize in the table
`dataElement`: Variable specifying the protection method

Returns

This UDF returns the tokenized value

Example

```
select pty_IntegerIns(column_name,'int_tk_de') from table_name;
```

8.3.16 pty_IntegerSel()

This UDF returns the detokenized value for a column containing *integer* format data.

pty_IntegerSel(col integer, dataElement varchar)

Parameters

`col`: Column name of the data to detokenize in the table
`dataElement`: Variable specifying the unprotection method

Returns

This UDF returns the detokenized value

Example

```
select pty_IntegerSel(column_name,'enc_de') from table_name;
```

8.3.17 pty_DateEnc()

This UDF returns the encrypted value for a column containing *date* format data.

pty_DateEnc(col date, dataElement varchar)

Parameters

`col`: Date column to encrypt in the table
`dataElement`: Variable specifying the protection method

Returns

This UDF returns the encrypted value as byte array

Example

```
select pty_DateEnc(column_name,'enc_de') from table_name;
```

8.3.18 pty_DateDec()

This UDF returns the decrypted value for a column containing encrypted data in *byte array* format.

pty_DateDec(col byte[], dataElement varchar)

Parameters

col: Date column to decrypt in the table

dataElement: Variable specifying the unprotection method

Returns

This UDF returns the decrypted value

Example

```
select pty_DateDec(column_name,'enc_de') from table_name;
```

8.3.19 pty_DateHash()

This UDF returns the hashed value for a column containing data in *date* format.

pty_DateHash(col date, dataElement varchar)

Parameters

col: Date column to hash in the table

dataElement: Variable specifying the protection method

Returns

This UDF returns the hashed value as byte array

Example

```
select pty_DateHash(column_name,'hash_de') from table_name;
```

8.3.20 pty_DateIns()

This UDF returns the tokenized value for a column containing data in *date* format.

pty_DateIns(col date, dataElement varchar)

Parameters

col: Date column to tokenize in the table

dataElement: Variable specifying the protection method

Returns

This UDF returns the tokenized value as date

If the date provided is out of the range than that described in *Protection Methods Reference 6.5.2*, then an error message appears in the *psql* shell and the transaction is aborted. An Audit log entry is not generated for this issue.

Example

```
select pty_DateIns(column_name,'date_tk_de') from table_name;
```

8.3.21 pty_DateSel()

This UDF returns the detokenized value for a column containing data in *date* format.

pty_DateSel(col date, dataElement varchar)

Parameters

col: Date column to detokenize in the table

dataElement: Variable specifying the unprotection method

Returns
This UDF returns the detokenized value as date

Example
```
select pty_DateSel(column_name,'date_tk_de') from table_name;
```

8.3.22 pty_RealEnc()

This UDF returns the encrypted value for a column containing data in *decimal* format.

pty_RealEnc(col real, dataElement varchar)

Parameters
`col`: Column name of the data to encrypt in the table
`dataElement`: Variable specifying the protection method

Returns
This UDF returns the encrypted value in byte array format

Example
```
select pty_RealEnc(column_name,'enc_de') from table_name;
```

8.3.23 pty_RealDec()

This UDF returns the decrypted value for a column containing encrypted data in *byte array* format.

pty_RealDec(col real, dataElement varchar)

Parameters
`col`: Column name of the data to decrypt in the table
`dataElement`: Variable specifying the unprotection method

Returns
This UDF returns the decrypted value in real format

Example
```
select pty_RealDec(column_name,'enc_de') from table_name;
```

8.3.24 pty_RealHash()

This UDF returns the hashed value for a column containing data in *real* format.

pty_RealHash(col real, dataElement varchar)

Parameters
`col`: Column name of the data to hash in the table
`dataElement`: Variable specifying the protection method

Returns
This UDF returns the hashed value as byte array

Example
```
select pty_RealHash(column_name,'hash_de') from table_name;
```

8.3.25 pty_RealIns()

This UDF returns the tokenized value for a column containing data in *real* format.

If a decimal value is used, then it is tokenized by first loading the *decimal* type column into a *varchar* type column and then using **pty_VarcharIns()** to tokenize this column.

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

pty_RealIns(col real, dataElement varchar)

Parameters

`col`: Column name of the data to tokenize in the table

`dataElement`: Variable specifying the protection method

Result

This UDF returns the tokenized value in real format

Example

```
select pty_RealIns(column_name,'noenc_de') from table_name;
```

8.3.26 pty_RealSel()

This UDF returns the detokenized value for a column containing data in *real* format.

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

If used with any other type of data, then an error explaining that the datatype is unsupported should be reported.

pty_RealSel(col real, dataElement varchar)

Parameters

`col`: Column name of the data to detokenize in the table

`dataElement`: Variable specifying the unprotection method

Returns

This UDF returns the detokenized value

Example

```
select pty_RealSel(column_name,'noenc_de') from table_name;
```

8.4 Inserting Data from a File into a Table

➤ **To populate the table *sample_table* from the *basic_sample_data.csv* file:**

The following command creates the table *sample_table*, with the required number of columns.

```
create table sample_table (colname1 colname1_format, colname2 colname2_format,
colname3 colname3_format) distributed randomly;
```

The following command grants permissions for the table *sample_table* to the required user, which will be used to perform the protect or unprotect operations.

```
grant all on sample_table to <username>;
```

The following command enables you to populate the table *sample_table* with the data from the *basic_sample_data.csv* file from the *<PROTEGRITY_DIR>/samples/data* directory.

```
\copy sample_table from '/opt/protegrity/samples/data/basic_sample_data.csv'
with delimiter ','
```

Parameters

sample_table: Name of the HAWQ table created to load the data from the input CSV file from the required path.

colname1, colname2, colname3: Name of the columns.

colname1_format, colname2_format, colname3_format: The data types contained in the respective columns. The data types can only be of types VARCHAR, INTEGER, DATE or REAL.

ATTRIBUTES: Additional parameters to consider when ingesting the data.

In the example, the row format is delimited using the ‘,’ character as the row format in the input file is comma separated. If the input file is tab separated, then the the row format is delimited using '\t'.

8.5 Protecting Existing Data

To protect existing data, users should define the mappings between the columns and their respective data elements in the data security policy.

The following commands create the table *basic_sample_protected* to store the protected data.

```
drop table if exists basic_sample_protected;

create table basic_sample_protected (colname1  colname1_format, colname2
colname2_format, colname3 colname3_format) distributed randomly;
```

Ensure that the user performing the task has the permissions to protect the data, as required, in the data security policy.

The following command ingests cleartext data from the *basic_sample* table to the *basic_sample_protected* table in protected form using HAWQ UDFs.

```
insert into basic_sample_protected(colname1, colname2, colname3) select colname1,
pty_varcharins(colname2,dataElement2), pty_varcharins(colname3,dataElement3) from
basic_sample;
```

Parameters

basic_sample_protected: Table to store protected data.

colname1, colname2, colname3: Name of the columns.

dataElement1, dataElement2, dataElement3: The data elements corresponding to the columns.

basic_sample: Table containing the original data in cleartext form.

8.6 Unprotecting Protected Data

To unprotect protected data, you need to specify the name of the table which contains the protected data, the table which would store the unprotected data, and the columns and their respective data elements.

Ensure that the user performing the task has permissions to unprotect the data as required in the data security policy.

The following commands create the table *basic_sample_unprotected* to store the unprotected data.

```
drop table if exists table_unprotected;

create table table_unprotected (colname1 colname1_format, colname2
colname2_format, colname3 colname3_format) distributed randomly;
```

The following command retrieves the unprotected data and saves it in the *basic_sample_unprotected* table.

```
insert into table_unprotected (colname1, colname2, colname3) select colname1,
pty_varcharsel(colname2,dataElement2), pty_varcharsel(colname3,dataElement3) from
table_protected;
```

Parameters

table_unprotected: Table to store unprotected data.

colname1, colname2, colname3: Name of the columns.

dataElement1, dataElement2, dataElement3: The data elements corresponding to the columns.

table_protected: Table containing protected data.

8.7 Retrieving Data from a Table

To retrieve data from a table, the user needs to have access to the table.

The following command displays the data contained in the table.

```
select * from table;
```

Parameters

table: Name of the table.

8.8 Sample Use Cases

For information about the HAWQ protector sample use cases, refer to section *11.10 Protecting Data using HAWQ*.

9 Spark

Spark is an execution engine that carries out batch processing of jobs in-memory and handles a wider range of computational workloads. In addition to processing a batch of stored data, Spark is capable of manipulating data in real time.

Spark leverages the physical memory of the Hadoop system and utilizes Resilient Distributed Datasets (RDDs) to store the data in-memory and lowers latency, if the data fits in the memory size. The data is saved on the hard drive only if required. As RDDs are the basic units of abstraction and computation in Spark, you can use the protection and unprotection APIs, provided by the Spark protector, when performing the transformation operations on an RDD.

If you need to use the Spark Protector API in a Spark Java job, then the users will have to implement the function interface as per the Spark Java programming specifications and subsequently use it in the required transformation of an RDD to tokenize the data.

This section provides information about the Spark protector, the APIs provided, and the commands for protecting and unprotecting data in a file by using the respective Spark APIs for protection or unprotection. In addition, it provides information about Spark SQL, which is a module that adds relational data processing capabilities to the Spark APIs, and a sample program for Spark Scala.

9.1 Overview of the Spark Protector

The Protegrity Spark protector extends the functionality of the Spark engine and provides APIs that protect or unprotect the data as it is stored or retrieved.

9.2 Spark Protector Usage

The Protegrity Spark protector provides APIs for protecting and reprotecting the data using encryption or tokenization, and unprotecting data by using decryption or detokenization.

Ensure that Spark is configured after the Big Data Protector is installed.

For more information about configuring Spark, refer to section *3.1.12 Configuring Spark.*

9.3 Spark APIs

This section describes the Spark APIs (Java) available for protection and unprotection in the Big Data Protector to build secure Big Data applications.

The Protegrity Spark protector only supports *bytes* converted from the *string* data type.

If *int*, *short*, or *long* format data is directly converted to *bytes* and passed as input to the API that supports *byte* as input and provides *byte* as output, then data corruption might occur.

9.3.1 getVersion()

This function returns the current version of the Spark protector.

public string getVersion()

Parameters

None

Result

This function returns the current version of Spark protector.

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector(applicationId);
String version = protector.getVersion();
```

Exception

PtySparkProtectorException: If unable to return the current version of the Spark protector

9.3.2 getCurrentKeyId()

This method returns the current Key ID for the data element, which contains the *KEY ID* attribute, while creating the data element, such as AES-256, AES-128, and so on.

public int getCurrentKeyId(String dataElement)

Parameters

`dataElement`: Name of the data element

Result

This method returns the current Key ID for the data element containing the *KEY ID* attribute.

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector(applicationId);
int keyId = protector.getCurrentKeyId("AES-256");
```

Exception

PtySparkProtectorException: If unable to return the current Key ID for the data element

9.3.3 checkAccess()

This method checks the access of the user for the specified data element.

public boolean checkAccess(String dataElement, Permission permission)

Parameters

`dataElement`: Name of the data element

`Permission`: Type of the access of the user for the data element

Result

`true`: If the user has access to the data element

`false`: If the user does not have access to the data element

Example

```
String applicationId = sparkContext.getConf().getAppId()
Protector protector = new PtySparkProtector(applicationId);
boolean accessType = protector.checkAccess(dataElement, Permission.PROTECT);
```

Exception

PtySparkProtectorException: If unable to verify the access of the user for the data element

9.3.4 getDefaultDataElement()

This method returns default data element configured in the security policy.

public String getDefaultDataElement(String policyName)

Parameters

`policyName`: Name of policy configured using Policy management in ESA

Result

Default data element name configured in the security policy.

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector(applicationId);
String dataElement = protector.getDefaultDataElement("sample_policy");
```

Exception

PtySparkProtectorException: If unable to return the default data element name

9.3.5 hmac()

This method performs hashing of the data using the HMAC operation on a single data item with a data element, which is associated with HMAC. It returns the hmac value of the data with the data element.

public byte[] hmac(String dataElement, byte[] input)

Parameters

`dataElement`: Name of the data element for HMAC

`data`: Byte *array* of data for HMAC

Result

Byte array of HMAC data

Example

```
String applicationId = sparkContext.getConf().getAppId()
Protector protector = new PtySparkProtector(applicationId);
byte[] output = protector.hmac("HMAC-SHA1", "test1".getBytes());
```

Exception

PtySparkProtectorException: If unable to protect data

9.3.6 protect()

Protects the data provided as a byte *array*. The type of protection applied is defined by *`dataElement`*.

public void protect(String dataElement, List<Integer> errorIndex, byte[][] input, byte[][] output)

Parameters

`dataElement`: Name of the data element used for protection

`errorIndex`: List of the Error Index

`input`: Array of a byte array of data to be protected

`output`: Array of a byte array containing protected data

The Protegrity Spark protector only supports *bytes* converted from the *string* data type.

If *int*, *short*, or *long* format data is directly converted to *bytes* and passed as input to the API that supports *byte* as input and provides *byte* as output, then data corruption might occur.

If you are using the *Protect* API which accepts *byte* as input and provides *byte* as output, then ensure that when unprotecting the data, the *Unprotect* API, with *byte* as input and *byte* as output is utilized. In addition, ensure that the *byte* data being provided as input to the Protect API has been converted from a *string* data type only.

Result

The *output* variable in the method signature contains protected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement=”Binary”;
byte[][] input = new byte[][]{“test1”.getbytes(),”test2”.getbytes()};
byte[][] output = new byte[input.length][];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to protect data

9.3.7 protect()

Protects the short format data provided as a short *array*. The type of protection applied is defined by `dataElement`.

public void protect(String dataElement, List<Integer> errorIndex, short[] input, short[] output)

Parameters

`dataElement`: Name of the data element used for protection

`errorIndex`: List of the Error Index

`input`: Short *array* of data to be protected

`output`: Short *array* containing protected data

Result

The *output* variable in the method signature contains protected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement=”short”;
short[] input = new short[] {1234, 4545};
short[] output = new short[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to protect data

9.3.8 protect()

Encrypts the short format data provided as a short *array*. The type of encryption applied is defined by *dataElement*.

public void protect(String dataElement, List<Integer> errorIndex, short[] input, byte[][] output)

Parameters

dataElement: Name of the data element used for encryption

errorIndex: List of the Error Index

input: Short *array* of data to be encrypted

output: Array of an encrypted byte *array*

Result

The *output* variable in the method signature contains protected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AES-256";
short[] input = new short[]{1234, 4545};
byte[][] output = new byte[input.length][];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to encrypt data

9.3.9 protect()

Protects the data provided as *int* array. The type of protection applied is defined by *dataElement*.

public void protect(String dataElement, List<Integer> errorIndex, int[] input, int[] output)

Parameters

dataElement: Name of the data element used for protection

errorIndex: List of the Error Index

input: Int *array* of data to be protected

output: Int *array* containing protected data

Result

The *output* variable in the method signature contains protected *int* data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "int";
int[] input = new int[]{1234, 4545};
int[] output = new int[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to protect data

9.3.10 protect()

Encrypts the data provided as *int* array. The type of encryption applied is defined by `dataElement`.

public void protect(String dataElement, List<Integer> errorIndex, int[] input, byte[][] output)

Parameters

`dataElement`: Name of the data element used for encryption

`errorIndex`: List of the Error Index

`input`: Int *array* of data to be encrypted

`output`: Array of an encrypted byte *array*

Result

The *output* variable in the method signature contains encrypted data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AES-256";
int[] input = new int[]{1234, 4545};
byte[][] output = new byte[input.length][];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to encrypt data

9.3.11 protect()

Protects the data provided as *long* byte array. The type of protection applied is defined by `dataElement`.

public void protect(String dataElement, List<Integer> errorIndex, long[] input, long[] output)

Parameters

`dataElement`: Name of the data element used for protection

`errorIndex`: List of the Error Index

`input`: Long *array* of data to be protected

`output`: Long *array* containing protected data

Result
The *output* variable in the method signature contains protected data

Example
```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "long";
long[] input = new long[] {1234, 4545};
long[] output = new long[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception
PtySparkProtectorException: If unable to protect data

9.3.12 protect()

Encrypts the data provided as *long* byte array. The type of encryption applied is defined by `dataElement`.

public void protect(String dataElement, List<Integer> errorIndex, long[] input, byte[][] output)

Parameters
`dataElement`: Name of the data element used for encryption
`errorIndex`: List of the Error Index
`input`: Long *array* of data to be encrypted
`output`: Array of a byte *array* containing encrypted data

Result
The *output* variable in the method signature contains encrypted data

Example
```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AES-256";
long[] input = new long[] {1234, 4545};
byte[][] output = new byte[input.length][];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception
PtySparkProtectorException: If unable to encrypt data

9.3.13 protect()

Protects the data provided as *float* array. The type of protection applied is defined by `dataElement`.

public void protect(String dataElement, List<Integer> errorIndex, float[] input, float[] output)

Parameters
`dataElement`: Name of the data element used for protection
`errorIndex`: List of the Error Index

`input`: Float *array* of data to be protected

`output`: Float *array* containing protected data

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

The *output* variable in the method signature contains protected *float* data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "float";
float[] input = new float[] {123.4f, 454.5f};
float[] output = new float[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to protect data

9.3.14 protect()

Encrypts the data provided as *float* array. The type of encryption applied is defined by *`dataElement`*.

public void protect(String dataElement, List<Integer> errorIndex, float[] input, byte[][] output)

Parameters

`dataElement`: Name of the data element used for encryption

`errorIndex`: List of the Error Index

`input`: Float *array* of data to be encrypted

`output`: Array of a byte *array* containing encrypted data

Ensure that you use the data element with either the *No Encryption* method or *Encryption* data element only. Using any other data element might cause corruption of data.

Result

The *output* variable in the method signature contains encrypted data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AES-256";
float[] input = new float[] {123.4f, 454.5f};
byte[][] output = new byte[input.length][];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to encrypt data

9.3.15 protect()

Protects the data provided as *double* array. The type of protection applied is defined by *`dataElement`*.

public void protect(String dataElement, List<Integer> errorIndex, double[] input, double[] output)

Parameters

`dataElement`: Name of the data element used for protection

`errorIndex`: List of the Error Index

`input`: Double *array* of data to be protected

`output`: Double *array* containing protected data

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

The *output* variable in the method signature contains protected *double* data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "double";
double[] input = new double[] {123.4, 454.5};
double[] output = new double[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to protect data

9.3.16 protect()

Encrypts the data provided as *double* array. The type of encryption applied is defined by *`dataElement`*.

public void protect(String dataElement, List<Integer> errorIndex, double[] input, byte[][] output)

Parameters

`dataElement`: Name of the data element used for encryption

`errorIndex`: List of the Error Index

`input`: Double *array* of data to be encrypted

`output`: Array of a byte *array* containing encrypted data

Ensure that you use the data element with either the *No Encryption* method or *Encryption* data element only. Using any other data element might cause corruption of data.

Result

The *output* variable in the method signature contains encrypted data

Example

```
   String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AES-256";
double[] input = new double[] {123.4, 454.5};
byte[][] output = new byte[input.length][];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to encrypt data

9.3.17 protect()

Protects the data provided as *string* array. The type of protection applied is defined by *dataElement*.

public void protect(String dataElement, List<Integer> errorIndex, String[] input, String[] output)

Parameters

`dataElement`: Name of the data element used for protection

`errorIndex`: List of the Error Index

`input`: String *array* of data to be protected

`output`: String *array* containing protected data

Result

The *output* variable in the method signature contains protected *string* data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AlphaNum";
String[] input = new String[] {"test1", "test2"};
String[] output = new String[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to protect data

9.3.18 protect()

Encrypts the data provided as *string* array. The type of encryption applied is defined by *dataElement*.

public void protect(String dataElement, List<Integer> errorIndex, String[] input, byte[][] output)

Parameters

`dataElement`: Name of the data element used for encryption

`errorIndex`: List of the Error Index

`input`: String *array* of data to be encrypted

`output`: Array of a byte *array* containing encrypted data

Result

The *output* variable in the method signature contains encrypted data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AES-256";
String[] input = new String[] {"test1", "test2"};
byte[][] output = new byte[input.length][];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.protect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to encrypt data

9.3.19 unprotect()

Unprotects the protected data provided as a byte *array*. The type of unprotection applied is defined by *dataElement*.

public void unprotect(String dataElement, List<Integer> errorIndex, byte[][] input, byte[][] output)

Parameters

`dataElement`: Name of the data element used for unprotection

`errorIndex`: List of the Error Index

`input`: Array of a byte *array* of data to be unprotected

`output`: Array of a byte *array* containing unprotected data

The Protegrity Spark protector only supports *bytes* converted from the *string* data type.

If *int*, *short*, or *long* format data is directly converted to *bytes* and passed as input to the API that supports *byte* as input and provides *byte* as output, then data corruption might occur.

Result

The *output* variable in the method signature unprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "Binary";
byte[][] input = new byte[][] {“test1”.getbytes(), ”test2”.getbytes()};
byte[][] output = new byte[input.length][];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to unprotect data

9.3.20 unprotect()

Unprotects the protected short format data provided as a short *array*. The type of unprotection applied is defined by *dataElement*.

public void unprotect(String dataElement, List<Integer> errorIndex, short[] input, short[] output)

Parameters

`dataElement`: Name of the data element used for unprotection

`errorIndex`: List of the Error Index

`input`: Short *array* of data to be unprotected

`output`: Short *array* containing unprotected data

Result

The *output* variable in the method signature contains unprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "short";
short[] input = new short[]{1234, 4545};
short[] output = new short[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to unprotect data

9.3.21 unprotect()

Decrypts the encrypted short format data provided as a byte *array*. The type of decryption applied is defined by *`dataElement`*.

public void unprotect(String dataElement, List<Integer> errorIndex, byte[][] input, short[] output)

Parameters

`dataElement`: Name of the data element used for decryption

`errorIndex`: List of the Error Index

`input`: Array of a byte *array* containing encrypted data

`output`: Short *array* containing decrypted data

Result

The *output* variable in the method signature contains decrypted data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AES-256";
// here input is encrypted short array created using our below API
// public void protect(String dataElement, List<Integer> errorIndex, short[]
      input, byte[][] output) throws PtySparkProtectorException;
byte[][] input = { <encrypted short array> }
short[] output = new short[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to decrypt data

9.3.22 unprotect()

Unprotects the protected data provided as *int* array. The type of unprotection applied is defined by *dataElement*.

public void unprotect(String dataElement, List<Integer> errorIndex, int[] input, int[] output)

Parameters

dataElement: Name of the data element used for unprotection

errorIndex: List of the Error Index

input: Int *array* of data to be unprotected

output: Int *array* containing unprotected data

Result

The *output* variable in the method signature contains unprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "int";
int[] input = new int[]{1234, 4545};
int[] output = new int[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to unprotect data

9.3.23 unprotect()

Decrypts the encrypted int format data provided as byte array. The type of decryption applied is defined by *dataElement*.

public void unprotect(String dataElement, List<Integer> errorIndex, byte[][] input, int[] output)

Parameters

dataElement: Name of the data element used for decryption

errorIndex: List of the Error Index

input: Array of a byte *array* containing encrypted data

output: Int *array* containing decrypted data

Result

The *output* variable in the method signature contains decrypted data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AES-256";
// here input is encrypted int array created using our below API
// public void protect(String dataElement, List<Integer> errorIndex, int[]
      input, byte[][] output) throws PtySparkProtectorException;
byte[][] input = {<encrypted int array>};
int[] output = new int[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to decrypt data

9.3.24 unprotect()

Unprotects the protected data provided as *long* array. The type of unprotection applied is defined by *dataElement*.

public void unprotect(String dataElement, List<Integer> errorIndex, long[] input, long[] output)

Parameters

dataElement: Name of the data element used for unprotection

errorIndex: List of the Error Index

input: Long *array* of data to be unprotected

output: Long *array* containing unprotected data

Result

The *output* variable in the method signature contains unprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "long";
long[] input = new long[] {1234, 4545};
long[] output = new long[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to unprotect data

9.3.25 unprotect()

Decrypts the encrypted long format data provided as byte array. The type of decryption applied is defined by *dataElement*.

public void unprotect(String dataElement, List<Integer> errorIndex, byte[][] input, long[] output)

Parameters

dataElement: Name of the data element used for decryption

errorIndex: List of the Error Index

`input`: Array of a byte *array* containing encrypted data

`output`: Long *array* containing decrypted data

Result

The *output* variable in the method signature decrypted data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AES-256";
// here input is encrypted long array created using our below API
// public void protect(String dataElement, List<Integer> errorIndex, long[]
      input, byte[][] output) throws PtySparkProtectorException;
byte[][] input = { <encrypted long array> };
long[] output = new long[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to unprotect data

9.3.26 unprotect()

Unprotects the protected data provided as *float* array. The type of unprotection applied is defined by *`dataElement`*.

public void unprotect(String dataElement, List<Integer> errorIndex, float[] input, float[] output)

Parameters

`dataElement`: Name of the data element used for unprotection

`errorIndex`: List of the Error Index

`input`: Float *array* of data to be unprotected

`output`: Float *array* containing unprotected data

Result

The *output* variable in the method signature contains unprotected data

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "float";
float[] input = new float[] {123.4f, 454.5f};
float[] output = new float[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to unprotect data

9.3.27 unprotect()

Decrypts the encrypted float format data provided as byte array. The type of decryption applied is defined by *dataElement*.

public void unprotect(String dataElement, List<Integer> errorIndex, byte[][] input, float[] output)

Parameters

dataElement: Name of the data element used for decryption

errorIndex: List of the Error Index

input: Array of a byte *array* containing encrypted data

output: Float *array* containing decrypted data

Ensure that you use the data element with either the *No Encryption* method or *Encryption* data element only. Using any other data element might cause corruption of data.

Result

The *output* variable in the method signature contains decrypted data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AES-256";
// here input is encrypted float array created using our below API
// public void protect(String dataElement, List<Integer> errorIndex, float[]
      input, byte[][] output) throws PtySparkProtectorException;
byte[][] input = { <encrypted float array> };
float[] output = new float[input.length][];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to decrypt data

9.3.28 unprotect()

Unprotects the protected data provided as *double* array. The type of unprotection applied is defined by *dataElement*.

public void unprotect(String dataElement, List<Integer> errorIndex, double[] input, double[] output)

Parameters

dataElement: Name of the data element used for unprotection

errorIndex: List of the Error Index

input: Double *array* of data to be unprotected

output: Double *array* containing unprotected data

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

The *output* variable in the method signature contains unprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "double";
double[] input = new double[] {123.4, 454.5};
double[] output = new double[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to unprotect data

9.3.29 unprotect()

Decrypts the encrypted double format data provided as byte array. The type of decryption applied is defined by *dataElement*.

public void unprotect(String dataElement, List<Integer> errorIndex, byte[][] input, double[] output)

Parameters

`dataElement`: Name of the data element used for decryption

`errorIndex`: List of the Error Index

`input`: Array of a byte *array* containing encrypted data

`output`: Double *array* containing decrypted data

Ensure that you use the data element with either the *No Encryption* method or *Encryption* data element only. Using any other data element might cause corruption of data.

Result

The *output* variable in the method signature contains decrypted data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AES-256";
// here input is encrypted double array created using our below API
// public void protect(String dataElement, List<Integer> errorIndex, double[]
      input, byte[][] output) throws PtySparkProtectorException;
byte[][] input = { <encrypted double array> };
double[] output = new double[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to unprotect data

9.3.30 unprotect()

Unprotects the protected data provided as *string* array. The type of unprotection applied is defined by *dataElement*.

public void unprotect(String dataElement, List<Integer> errorIndex, String[] input, String[] output)

Parameters

`dataElement`: Name of the data element used for unprotection

`errorIndex`: List of the Error Index

`input`: String *array* of data to be unprotected

`output`: String *array* containing unprotected data

Result

The *output* variable in the method signature contains unprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AlphaNum";
String[] input = new String[] {"test1", "test2"};
String[] output = new String[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to unprotect data

9.3.31 unprotect()

Decrypts the encrypted string format data provided as byte array. The type of decryption applied is defined by *`dataElement`*.

public void unprotect(String dataElement, List<Integer> errorIndex, byte[][] input, String[] output)

Parameters

`dataElement`: Name of the data element used for decryption

`errorIndex`: List of the Error Index

`input`: Array of a byte *array* containing encrypted data

`output`: String *array* containing decrypted data

Result

The *output* variable in the method signature contains decrypted data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String dataElement = "AES-256";
// here input is encrypted String array created using our below API
// public void protect(String dataElement, List<Integer> errorIndex, String[]
      input, byte[][] output) throws PtySparkProtectorException;
byte[][] input = { <encrypted string array> };
String[] output = new String[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.unprotect(dataElement, errorIndexList, input, output);
```

Exception

PtySparkProtectorException: If unable to decrypt data

9.3.32 reprotect()

Reprotects the byte *array* data, which was protected earlier, with a different data element.

public void reprotect(String oldDataElement, String newDataElement, List<Integer> errorIndex, byte[][] input, byte[][] output)

Parameters

`oldDataElement`: Name of the data element with which data was protected earlier

`newDataElement`: Name of the new data element with which data is reprotected

`errorIndex`: List of the Error Index

`input`: Array of a byte *array* of data to be reprotected

`output`: Array of a byte *array* containing reprotected data

Result

The *output* variable in the method signature contains reprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String oldDataElement = "Binary";
String newDataElement = "Binary_1";
byte[][] input = new byte[][] {"test1".getBytes(), "test2".getBytes()};
byte[][] output = new byte[input.length][];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.reprotect(oldDataElement, newDataElement, errorIndexList, input,
      output);
```

Exception

PtySparkProtectorException: If errors occur while reprotecting data

9.3.33 reprotect()

Reprotects the short *array* data, which was protected earlier, with a different data element.

public void reprotect(String oldDataElement, String newDataElement, List<Integer> errorIndex, short[] input, short[] output)

Parameters

`oldDataElement`: Name of the data element with which data was protected earlier

`newDataElement`: Name of the new data element with which data is reprotected

`errorIndex`: List of the Error Index

`input`: Short *array* of data to be reprotected

`output`: Short *array* containing reprotected data

Result

The *output* variable in the method signature contains reprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String oldDataElement = "short";
String newDataElement = "short_1";
short[] input = new short[] {135, 136};
short[] output = new short[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.reprotect(oldDataElement, newDataElement, errorIndexList, input,
      output);
```

Exception

PtySparkProtectorException: If errors occur while reprotecting data

9.3.34 reprotect()

Reprotects the int *array* data, which was protected earlier, with a different data element.

public void reprotect(String oldDataElement, String newDataElement, List<Integer> errorIndex, int[] input, int[] output)

Parameters

`oldDataElement`: Name of the data element with which data was protected earlier

`newDataElement`: Name of the new data element with which data is reprotected

`errorIndex`: List of the Error Index

`input`: Int *array* of data to be reprotected

`output`: Int *array* containing reprotected data

Result

The *output* variable in the method signature contains reprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String oldDataElement = "int";
String newDataElement = "int_1";
int[] input = new int[] {234,351};
int[] output = new int[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.reprotect(oldDataElement, newDataElement, errorIndexList, input,
      output);
```

Exception

PtySparkProtectorException: If errors occur while reprotecting data

9.3.35 reprotect()

Reprotects the long *array* data, which was protected earlier, with a different data element.

public void reprotect(String oldDataElement, String newDataElement, List<Integer> errorIndex, long[] input, long[] output)

Parameters

`oldDataElement`: Name of the data element with which data was protected earlier

`newDataElement`: Name of the new data element with which data is reprotected

`errorIndex`: List of the Error Index

`input`: Long *array* of data to be reprotected

`output`: Long *array* containing reprotected data

Result

The *output* variable in the method signature contains reprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String oldDataElement = "long";
String newDataElement = "long_1";
long[] input = new long[] {1234, 135}};
long[] output = new long[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.reprotect(oldDataElement, newDataElement, errorIndexList, input,
      output);
```

Exception

PtySparkProtectorException: If errors occur while reprotecting data

9.3.36 reprotect()

Reprotects the float *array* data, which was protected earlier, with a different data element.

public void reprotect(String oldDataElement, String newDataElement, List<Integer> errorIndex, float[] input, float[] output)

Parameters

`oldDataElement`: Name of the data element with which data was protected earlier

`newDataElement`: Name of the new data element with which data is reprotected

`errorIndex`: List of the Error Index

`input`: Float *array* of data to be reprotected

`output`: Float *array* containing reprotected data

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

The *output* variable in the method signature contains reprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String oldDataElement = "NoEnc";
String newDataElement = "NoEnc_1";
float[] input = new float[] {23.56f, 26.43f}};
float[] output = new float[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.reprotect(oldDataElement, newDataElement, errorIndexList, input,
      output);
```

Exception

PtySparkProtectorException: If errors occur while reprotecting data

9.3.37 reprotect()

Reprotects the double *array* data, which was protected earlier, with a different data element.

public void reprotect(String oldDataElement, String newDataElement, List<Integer> errorIndex, double[] input, double[] output)

Parameters

`oldDataElement`: Name of the data element with which data was protected earlier

`newDataElement`: Name of the new data element with which data is reprotected

`errorIndex`: List of the Error Index

`input`: Double *array* of data to be reprotected

`output`: Double *array* containing reprotected data

Ensure that you use the data element with the *No Encryption* method only. Using any other data element might cause corruption of data.

Result

The *output* variable in the method signature contains reprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String oldDataElement = "NoEnc";
String newDataElement = "NoEnc_1";
double[] input = new double[] {235.5, 1235.66};
double[] output = new double[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.reprotect(oldDataElement, newDataElement, errorIndexList, input,
      output);
```

Exception

PtySparkProtectorException: If errors occur while reprotecting data

9.3.38 reprotect()

Reprotects the string *array* data, which was protected earlier, with a different data element.

public void reprotect(String oldDataElement, String newDataElement, List<Integer> errorIndex, String[] input, String[] output)

Parameters

`oldDataElement`: Name of the data element with which data was protected earlier

`newDataElement`: Name of the new data element with which data is reprotected

`errorIndex`: List of the Error Index

`input`: String *array* of data to be reprotected

`output`: String *array* containing reprotected data

Result

The *output* variable in the method signature contains reprotected data

Example

```
String applicationId = sparkContext.getConf().getAppId();
Protector protector = new PtySparkProtector (applicationId);
String oldDataElement = "AlphaNum";
```

```
String newDataElement = "AlphaNum_1";
String[] input = new String[] {"test1", "test2"};
String[] output = new String[input.length];
List<Integer> errorIndexList = new ArrayList<Integer>();
protector.reprotect(oldDataElement, newDataElement, errorIndexList, input,
      output);
```

Exception

PtySparkProtectorException: If errors occur while reprotecting data

9.4 Displaying the Cleartext Data from a File

➤ **To display the cleartext data from the *basic_sample_data.csv* file:**

The following command enables you to display the cleartext data from the *basic_sample_data.csv* file from the */tmp/basic_sample/sample/* directory.

```
hadoop fs -cat /tmp/basic_sample/sample/basic_sample_data.csv
```

Parameters

basic_sample_data.csv: Name of the file containing cleartext data.

9.5 Protecting Existing Data

To protect cleartext data, you need to specify the name of the file which contains the cleartext data and the name of the file which would store the protected data.

The following command reads the cleartext data from the *basic_sample_data.csv* file and stores it in the *basic_sample_protected.csv* file in protected form using Spark UDFs.

Ensure that the user performing the task has the permissions to protect the data, as required, in the data security policy.

```
./spark-submit --master yarn --class com.protegrity.spark.ProtectData
<PROTEGRITY_DIR>/samples/spark/lib/spark_protector_demo.jar

<Path_of_Cleartext_data_file>/basic_sample_data.csv

<Path_of_Protected_data_file>/basic_sample_protected.csv
```

Parameters

com.protegrity.spark.ProtectData: The Spark protector class for protecting the data.

spark_protector_demo.jar: The Jar file utilizing the Spark protector API for protecting data in the *.csv* file.

<Path_of_Cleartext_data_file>: The HDFS directory path for the file with cleartext data.

<Path_of_Protected_data_file>: The HDFS directory path for the file with protected data.

basic_sample_data: File to read cleartext data.

basic_sample_protected_data: File to store protected data.

9.6 Unprotecting Protected Data

To unprotect the protected data, you need to specify the name of the file which contains the protected data and the name of the file which would store the unprotected data.

The following command retrieves the protected data from the *basic_sample_protected.csv* file and saves it in *basic_sample_unprotected.csv* file in unprotected form.

Ensure that the user performing the task has the permissions to unprotect the data, as required, in the data security policy.

```
./spark-submit --master yarn --class com.protegrity.spark.UnProtectData
<PROTEGRITY_DIR>/samples/spark/lib/spark_protector_demo.jar

<Path_of_Protected_data_file>/basic_sample_protected_data.csv

<Path_of_Unprotected_data_file>/basic_sample_unprotected_data.csv
```

Parameters

com.protegrity.spark.UnProtectData: The Spark protector class for unprotecting the protected data.

spark_protector_demo.jar: The Jar file utilizing the Spark protector API for unprotecting the protected data in the *.csv* file.

<Path_of_Protected_data_file>: The HDFS directory path for the file with protected data.

<Path_of_Unprotected_data_file>: The HDFS directory path for the file to store the unprotected data.

basic_sample_protected_data: File to read the protected data.

basic_sample_unprotected_data: File to save the unprotected data.

9.7 Retrieving the Unprotected Data from a File

To retrieve data from a file containing protected data, the user needs to have access to the file.

The following command displays the unprotected data contained in the file.

```
hadoop fs -cat /tmp/basic_sample/sample/basic_sample_unprotected_data.csv/part*
```

Parameters

basic_sample_unprotected_data.csv: Name of the file containing unprotected data.

9.8 Spark APIs and Supported Protection Methods

The following table lists the Spark APIs, the input and output data types, and the supported Protection Methods.

Table 9-1: Spark APIs and Supported Protection Methods

Operation	Input	Output	Protection Method Supported
Protect	Byte	Byte	Tokenization, Encryption, No Encyption, DTP2, CUSP
Protect	Short	Short	Tokenization, No Encyption
Protect	Short	Byte	Encryption, CUSP
Protect	Int	Int	Tokenization, No Encyption
Protect	Int	Byte	Encryption, CUSP
Protect	Long	Long	Tokenization, No Encyption
Protect	Long	Byte	Encryption, CUSP
Protect	Float	Float	Tokenization, No Encyption
Protect	Float	Byte	Encryption, CUSP
Protect	Double	Double	Tokenization, No Encyption
Protect	Double	Byte	Encryption, CUSP
Protect	String	String	Tokenization, No Encyption, DTP2

Operation	*Input*	*Output*	*Protection Method Supported*
Protect	String	Byte	Encryption, CUSP
Unprotect	Byte	Byte	Tokenization, Encryption, No Encyption, DTP2, CUSP
Unprotect	Short	Short	Tokenization, NoEncyption
Unprotect	Byte	Short	Encryption, CUSP
Unprotect	Int	Int	Tokenization, No Encyption
Unprotect	Byte	Int	Encryption, CUSP
Unprotect	Long	Long	Tokenization, No Encyption
Unprotect	Byte	Long	Encryption, CUSP
Unprotect	Float	Float	Tokenization, No Encyption
Unprotect	Byte	Float	Encryption, CUSP
Unprotect	Double	Double	Tokenization, No Encyption
Unprotect	Byte	Double	Encryption, CUSP
Unprotect	String	String	Tokenization, No Encyption, DTP2
Unprotect	Byte	String	Encryption, CUSP
Reprotect	Byte	Byte	Tokenization, Encryption, DTP2, CUSP NOTE: If a protected value is generated using *Byte* as both *Input* and *Output*, then only Encryption/CUSP is supported.
Reprotect	Short	Short	Tokenization
Reprotect	Int	Int	Tokenization
Reprotect	Long	Long	Tokenization
Reprotect	Float	Float	Tokenization
Reprotect	Double	Double	Tokenization
Reprotect	String	String	Tokenization, DTP2

9.9 Sample Use Cases

For information about the Spark protector sample use cases, refer to section *12.11 Protecting Data using Spark*.

9.10 Spark SQL

The Spark SQL module provides relational data processing capabilities to Spark. The module allows you to run SQL queries with Spark programs. It contains DataFrames, which is an RDD with an associated schema, that provide support for processing structured data in Hive tables.

Spark SQL enables structured data processing and programming of RDDs providing relational and procedural processing through a DataFrame API that integrates with Spark.

Starting from the 6.6.3 release, you can invoke Protegrity Hive UDFs using Spark SQL and protect or unprotect data.

Starting from the Big Data Protector 6.6.3 release, the Spark SQL CLI is only supported.

9.10.1 DataFrames

A DataFrame is a distributed collection of data, such as RDDs, with a corresponding schema. DataFrames can be created from a wide array of sources, such as Hive tables, external databases, structured data files, or existing RDDs.

It can act as a distributed SQL query engine and is equivalent to a table in a relational database that can be manipulated, similar to RDDs. To optimize execution, DataFrames support relational operations and track their schema.

9.10.2 SQLContext

A SQLContext is a class that is used to initialize Spark SQL. It enables applications to run SQL queries, while running SQL functions, and provides the result as a DataFrame.

HiveContext extends the functionality of SQLContext and provides capabilities to use Hive UDFs, create Hive queries, and access and modify the data in Hive tables.

The Spark SQL CLI is used to run the Hive metastore service in local mode and execute queries. When we run Spark SQL (*spark-sql*), which is client for running queries in Spark, it creates a *SparkContext* defined as *sc* and *HiveContext* defined as *sqlContext*.

9.10.3 Accessing the Hive Protector UDFs

You can access Hive UDFs in Spark SQL by configuring the following Hive UDF JAR path in the Spark classpath.

For more information about configuring Spark SQL, refer to section *3.1.12 Configuring Spark.*

If you are using the Hive UDFs *ptyWhoAmI()* and *ptyProtectStr()* with Spark SQL, then the user retrieval process differs, as described by the following list:

- *ptyWhoAmI()*: The user is detected from the *SessionState* object, similar to Hive
- *ptyProtectStr()*: The user is detected from the *HiveConf* object as the *SessionState* object does not return the user.

  If you are using the UDF *ptyProtectStr()* with Hive, then the user is detected from the *SessionState* object.

9.10.3.1 Accessing Hive UDFs when Hive Services are Running

If the *hive-site.xml* file is placed in the Spark protector configuration directory, which is typically */etc/spark/conf*, then it can access the Hive UDFs, Hive tables, and read and write data that is stored data in Hive.

For more information about the Hive UDFs, refer to section *4.5 Hive UDFs*.

➤ To access Hive UDFs when Hive Services are running:

1. Ensure that Spark SQL is configured, as required.
   For more information about configuring Spark SQL, refer to section *3.1.12 Configuring Spark.*
2. Start the Spark SQL (*spark-sql*) client.
3. Create a table *test_table* with the following command.
   ```
   CREATE TABLE IF NOT EXISTS test_table (val STRING);
   ```

4. Load data from the *sample.txt* file in the table *test_table* with the following command.
   ```
   LOAD DATA LOCAL INPATH 'sample.txt' INTO TABLE test_table;
   ```
5. Protect the data in the table *test_table* using the Hive UDFs with the followind commands.
   ```
   create temporary function ptyProtectStr
   AS 'com.protegrity.hive.udf.ptyProtectStr';
   SELECT ptyProtectStr(val,'AN_TOKEN') from test_table
   ```
   The required data is protected by the data element.

9.10.3.2 Accessing Hive UDFs when Hive Services are Stopped

Spark SQL can access the Hive UDFs without Hive support when the Hive services are stopped.

When we run Spark SQL (*spark-sql*), it creates its own metastore (*metastore_db* file) and warehouse in the installation directory of Spark. Internally, Spark SQL compiles with Hive and uses the classes for internal execution of the Hive UDFs.

For more information about the Hive UDFs, refer to section *4.5 Hive UDFs*.

➤ **To access Hive UDFs when Hive Services are stopped:**

1. Ensure that the Hadoop cluster is installed, configured, and running.
2. Start the Spark SQL (*spark-sql*) client.
3. Create a table *test_table* with the following command.
   ```
   CREATE TABLE IF NOT EXISTS test_table (val STRING);
   ```
4. Load data from the *sample.txt* file in the table *test_table* with the following command.
   ```
   LOAD DATA LOCAL INPATH 'sample.txt' INTO TABLE test_table;
   ```
5. Protect the data in the table *test_table* using the Hive UDFs with the followind commands.
   ```
   create temporary function ptyProtectStr
   AS 'com.protegrity.hive.udf.ptyProtectStr';
   SELECT ptyProtectStr(val,'AN_TOKEN') from test_table
   ```
   The required data is protected by the data element.

9.10.4 Sample Use Cases

If you need to use Hive UDFs with Spark SQL, then initiate the Spark SQL (*spark-sql*) client. You can then specify commands which are similar to using Hive UDFs.

For information about the sample use cases for Hive UDFs using Spark SQL, refer to section *12.6 Protecting Data using Hive*.

9.11 Spark Scala

The Protegrity Spark protector (Java) can be used with Scala to protect and reprotect the data by using encryption or tokenization, and unprotect the data by using decryption or detokenization.

In this Big Data Protector release, a sample code snippet for Spark Scala is provided.

9.11.1 Sample Use Cases

For information about the sample code snippets for using Spark with Scala, refer to section *12.11.4 Sample Code Usage for Spark (Scala)*.

10 Data Node and Name Node Security with File Protector

The HDFS file system stores data in a distributed manner, which is accessible by any node that is a part of the Hadoop system.

The Hadoop utility uses commands to display and manipulate data in the HDFS file system. Hadoop stores information, such as blocks, metadata, jobs and so on in the OS file system. The Hadoop local directories are accessible to the OS user.

Protegrity File Protector over HDFS provides the way to protect the Hadoop local folder, which is stored locally in the operating system. The user will not be able to manipulate data or corrupt the folders. Any operation like read, write, delete and so on in the HDFS file system can be performed by users having access to the directories, only through binaries, such as hadoop, mapred, hive, pig and so on.

For managing access, Protegrity provides Policy management in ESA, which creates and deploys policies on Hadoop nodes.

For more information about Protegrity File Protector, contact Protegrity Professional Services.

10.1 Features of the Protegrity File Protector

You can use Protegrity File Protector to protect files and folders on the local file system.

The Protegrity File Protector provides a highly transparent and easy to administer solution for securing sensitive files and ensures the safety of your data, as explained in the following sections.

10.1.1 Protegrity File Encryption

Protegrity File Encryption protects file content in a transparent manner by encryption and decryption. This is accomplished by employing policies defined using Policy management in ESA and volume encryption modules.

10.1.2 Protegrity Volume Encryption

Protegrity Volume Encryption protects the contents of files and folders stored in a protected or encrypted volume employing policies defined using Policy management in ESA. The Administrator must mount the volume for use and then unmount and close it.

10.1.3 Protegrity Access Control

Protegrity Access Control protects directories, their child files and subdirectories in real-time from unauthorized file deletion, modification or read. This is accomplished by employing policies defined using Policy management in ESA and binding individual files and folders with their respective policies.

11 Appendix: Return Codes

If you are using the HDFSFP protector and any failures occur, then the protector thows an exception. The exception consists of an error code and error message. The following table lists all possible error codes and error descriptions.

Table 11-1 Error Codes for HDFSFP Protector

Code	Error	Error Description
0	XC_FAILED	The requested operation or service failed.
3	XC_LOG_CHECK_ACCESS	The access was denied as the user does not have the required privileges to perform the requested operation.
4	XC_LOG_TIME_ACCESS	The access was denied as the user does not have the required privileges to perform the requested operation at this point in time.
6	XC_LOG_ENCRYPT_SUCCESS	The data was successfully encrypted.
7	XC_LOG_ENCRYPT_FAILED	The encryption of data failed.
8	XC_LOG_DECRYPT_SUCCESS	The data was successfully decrypted.
9	XC_LOG_DECRYPT_FAILED	The decryption of data failed.
100	XC_INVALID_PARAMETER	The parameter specified in the function call is invalid.
101	XC_TIMEOUT	The operation timed out before a result was returned.
102	XC_ACCESS_DENIED	Permission to access an object or file on the filesystem is denied.
103	XC_NOT_SUPPORTED	The requested operation is not supported.
104	XC_SESSION_REFUSED	The remote peer client did not accept the session request.
105	XC_DISCONNECTED	The session was terminated.
106	XC_UNREACHABLE	The host could not be reached.
107	XC_SESSION_IN_USE	The session is already in use.
108	XC_EOF	The end of file is reached.
109	XC_NOT_FOUND	Not found.
110	XC_BUFFER_TOO_SMALL	Supplied input or output buffer is too small.

If you are using MapReduce, Hive, Pig, HBase, or Spark, and any failures occur, then the protector throws an exception. The exception consists of an error code and error message. The following table lists all possible error codes and error descriptions.

The following table lists all possible return codes provided to the PEP log files.

Table 11-2 PEP Log Return Codes

Code	Error	Error Description
0	NONE	
1	USER_NOT_FOUND	The user name could not be found in the policy residing in the shared memory.
2	DATA_ELEMENT_NOT_FOUND	The data element could not be found in the policy residing in the shared memory.
3	PERMISSION_DENIED	The user does not have the required permissions to perform the requested operation.

Code	Error	Error Description
4	TIME_PERMISSION_DENIED	The user does not have the appropriate permissions to perform the requested operation at this point in time.
5	INTEGRITY_CHECK_FAILED	Integrity check failed.
6	PROTECT_SUCCESS	The operation to protect the data was successful.
7	PROTECT_FAILED	The operation to protect the data failed.
8	UNPROTECT_SUCCESS	The operation to unprotect the data was successful.
9	UNPROTECT_FAILED	The operation to unprotect the data failed.
10	OK_ACCESS	The user has the required permissions to perform the requested operation. This return code ensures a verification and no data is protected or unprotected.
11	INACTIVE_KEYID_USED	The operation to unprotect the data was successful using an inactive Key ID.
12	INVALID_PARAM	The input is null or not within allowed limits.
13	INTERNAL_ERROR	An internal error occurred in a function call after the PEP provider is started.
14	LOAD_KEY_FAILED	Failed to load the data encryption key.
17	INIT_FAILED	The PEP server failed to initialize, which is a fatal error.
20	OUT_OF_MEMORY	Failed to allocate memory.
21	BUFFER_TOO_SMALL	The input or output buffer is very small.
22	INPUT_TOO_SHORT	The data is too short to be protected or unprotected.
23	INPUT_TOO_LONG	The data is too long to be protected or unprotected.
25	USERNAME_TOO_LONG	The user name is longer than the maximum supported length of the user name that can be used for protect or unprotect operations.
26	UNSUPPORTED	The algorithm or action for the specific data element is unsupported.
27	APPLICATION_AUTHORIZED	The application is authorized.
28	APPLICATION_NOT_AUTHORIZED	The application is not authorized.
29	JSON_NOTSERIALIZABLE	The JSON type is not serializable.
30	JSON_MALLOC_FAILED	The memory allocation for the JSON type failed.
31	EMPTY_POLICY	The policy residing in the shared memory is empty.
32	DELETE_SUCCESS	The operation to delete the data was successful.
33	DELETE_FAILED	The operation to delete the data failed.
34	CREATE_SUCCESS	The operation to create or add the data was successful.
35	CREATE_FAILED	The operation to create or add the data failed.
36	MNGPROT_SUCCESS	The management of the protection operation was successful.
37	MNGPROT_FAILED	The management of the protection operation failed.
39	POLICY_LOCKED	The policy residing in the shared memory is locked. This error can be caused by a *Disk Full* alert.

Code	Error	Error Description
40	LICENSE_EXPIRED	The license is not valid or the current date is beyond the license expiration date.
41	METHOD_RESTRICTED	The use of the Protection method is restricted by license.
42	LICENSE_INVALID	The license is invalid or the time is prior to the start of the license tenure.
44	INVALID_FORMAT	The content of the input data is invalid.
45	PROCESSING_SUCCESS	It is used for audit entries used for collecting Access Counter records.
46	INVALID_POLICY	It is used for a z/OS Query regarding the default data element when the policy name is not found.

The following table lists all possible result codes provided as a result of operations performed on the PEP.

Table 11-3 PEP Result Codes

Code	Error	Error Description
1	SUCCESS	The operation was successful.
0	FAILED	The operation failed.
-1	INVALID_PARAMETER	The parameter is invalid.
-2	EOF	The end of file was reached.
-3	BUSY	The operation is already in progress or the PEP server is busy with some other operation.
-4	TIMEOUT	The time-out threshold was reached as the PEP server was waiting for a response.
-5	ALREADY_EXISTS	The object, such as file, already exists.
-6	ACCESS_DENIED	The permission to access the object was denied.
-7	PARSE_ERROR	The error occurred when the contents were parsed.
-8	NOT_FOUND	The search operation was not successful.
-9	NOT_SUPPORTED	The operation is not supported.
-10	CONNECTION_REFUSED	The connection was refused.
-11	DISCONNECTED	The connection was terminated.
-12	UNREACHABLE	The Internet link is down or the host is not reachable.
-13	ADDRESS_IN_USE	The IP Adddress or port is already utilized.
-14	OUT_OF_MEMORY	The operation to allocate memory failed.
-15	CRC_ERROR	The CRC check failed.
-16	BUFFER_TOO_SMALL	The buffer size is very small.
-17	BAD_REQUEST	The message received was not in a standard format.
-18	INVALID_STRING_LENGTH	The input string is very long.
-19	INVALID_TYPE	The incorrect type of <NEED INPUTS> was used.
-20	READONLY_OBJECT	The object is set with read-only access.
-21	SERVICE_FAILED	The service failed.
-22	ALREADY_CONNECTED	The Administrator is already connected to the server.

Code	Error	Error Description
-23	INVALID_KEY	The key is invalid.
-24	INTEGRITY_ERROR	The integrity check failed.
-25	LOGIN_FAILED	The attempt to login failed.
-26	NOT_AVAILABLE	The object is not available.
-27	NOT_EXIST	The object does not exist.
-28	SET_FAILED	The Set operation failed.
-29	GET_FAILED	The Get operation failed.
-30	READ_FAILED	The Read operation failed.
-31	WRITE_FAILED	The Write operation failed.
-33	REWRITE_FAILED	The Rewrite operation failed.
-34	DELETE_FAILED	The Delete operation failed.
-35	UPDATE_FAILED	The Update operation failed.
-36	SIGN_FAILED	The Sign operation failed.
-37	VERIFY_FAILED	The Verification failed.
-38	ENCRYPT_FAILED	The Encrypt operation failed.
-39	DECRYPT_FAILED	The Decrypt operation failed.
-40	REENCRYPT_FAILED	The Reencrypt operation failed.
-41	EXPIRED	The object has expired.
-42	REVOKED	The object has been revoked.
-43	INVALID_FORMAT	The format is invalid.
-44	HASH_FAILED	The Hash operation failed.
-45	NOT_DEFINED	The property or setting is not defined.
-46	NOT_INITIALIZED	The service requested or function is performed on an object that is not initialized.
-47	POLICY_LOCKED	The Policy is locked.
-48	THROW_EXCEPTION	The error message is used to convey that an exception should be thrown during decryption.
-49	USER_AUTHENTICATION_FAILED	The Authentication operation failed.
-54	INVALID_CARD_TYPE	The credit card number provided does not confirm to the required credit card format.
-55	LICENSE_AUDITONLY	The License provided is for the audit functionality and only *No Encryption* data elements are allowed.
-56	NO_VALID_CIPHERS	No valid ciphers were found.
-57	NO_VALID_PROTOCOLS	No valid protocols were found.
-201	CRYPT_KEY_DATA_ILLEGAL	The key data specified is invalid.
-202	CRYPT_INTEGRITY_ERROR	The integrity check for the data failed.
-203	CRYPT_DATA_LEN_ILLEGAL	The data length specified is invalid.
-204	CRYPT_LOGIN_FAILURE	The Crypto login failed.
-205	CRYPT_CONTEXT_IN_USE	An attempt to close a key being used is made.

Code	Error	Error Description
-206	CRYPT_NO_TOKEN	The hardware token is available.
-207	CRYPT_OBJECT_EXISTS	The object to be created already exists.
-208	CRYPT_OBJECT_MISSING	A request for a non-existing object is made.
-221	X509_SET_DATA	The operation to set data in the object failed.
-222	X509_GET_DATA	The operation to get data from the object failed.
-223	X509_SIGN_OBJECT	The operation to sign the object failed.
-224	X509_VERIFY_OBJECT	The verification operation for the object failed.
-231	SSL_CERT_EXPIRED	The certificate has expired.
-232	SSL_CERT_REVOKED	The certificate has been revoked.
-233	SSL_CERT_UNKNOWN	The Trusted certificate was not found.
-234	SSL_CERT_VERIFY_FAILED	The certificate cound not be verified.
-235	SSL_FAILED	A general SSL error occurs.
-241	KEY_ID_FORMAT_ERROR	The format on the Key ID is invalid.
-242	KEY_CLASS_FORMAT_ERROR	The format on the KeyClass is invalid.
-243	KEY_EXPIRED	The key expired.
-250	FIPS_MODE_FAILED	The FIPS mode failed.

12 Appendix: Samples

Many organizations are adopting Hadoop due to its ability to process and analyze large volumes of unstructured data in a distributed manner. However, this might make sensitive data vulnerable with exposure to unauthorized users.

The Big Data Protector provides Hadoop users with top-to-bottom data protection from the application level to the file level. Sensitive data can be protected from internal and external threats and unauthorized users. The protected data can be utilized by users, business processes, and applications. In addition, the data is viewed in cleartext form by authorized users and in protected form by unauthorized users.

In the samples provided, data protection is done by tokenization, where sensitive data is converted to similar looking inert data known as tokens and the data format and type can be preserved. These tokens can be detokenized back to the original values when required.

The sample outputs provided in the documentation are for reference only. The values on your systems might vary than the ones listed in the document.

For ease of illustration, the following items have been added to documentation:

- **Protected data** in the samples is identified in dark grey color.
- Header rows in the output are retained to list the type of information contained in them. The actual output would not contain these header rows.
- Spaces are inserted in the output to demarcate data in the columns.

This section provides documentation for sample data protection for MapReduce, Hive, Pig, HBase, Impala, HAWQ, and Spark using Big Data Protector.

Ensure that you login as the user *root* before performing the following tasks.

➤ To copy the sample data:

This command copies the sample data from the installation directory to the */tmp/basic_sample/sample* directory in HDFS.

```
#> hadoop fs -rm -r /tmp/basic_sample/sample/
#> hadoop fs -rm -r /tmp/basic_sample
#> hadoop fs -mkdir /tmp/basic_sample/
#> hadoop fs -mkdir /tmp/basic_sample/sample/
#> hadoop fs -copyFromLocal /opt/protegrity/samples/data/basic_sample_data.csv
/tmp/basic_sample/sample
```

➤ To assign user permissions for the sample data in the Hadoop directories:

This command assigns permissions for the sample data in the Hadoop directories to all users.

```
#> sudo -u hdfs hadoop fs -chmod -R 777 /tmp/basic_sample
#> sudo -u hdfs hadoop fs -chmod -R 777 /apps/hive/warehouse
```

➤ To create the user John:

This command creates the user John.

```
#> useradd John
```

Ensure that you set a password for the user John using the command `passwd John`.

➤ To create the user Fred:

This command creates the user Fred.

```
#> useradd Fred
```

Ensure that you set a password for the user Fred using the command `passwd Fred`.

➤ **To create directories for the users John and Fred:**

This command creates the required directories for the users John and Fred in HDFS.

```
# sudo -u hdfs hadoop fs -mkdir /user/John
# sudo -u hdfs hadoop fs -chown -R John /user/John
# sudo -u hdfs hadoop fs -mkdir /user/Fred
# sudo -u hdfs hadoop fs -chown -R Fred /user/Fred
```

12.1 Roles in the Samples

The roles available in the samples are described in the following table.

Table 11-1: List of Roles

Roles	User	Role Description
SAMPLE_ADMIN	*root*	This user is able to protect and unprotect the data, and access the data in the cleartext form.
SAMPLE_INGESTION_USER	*John*	This user is allowed to ingest the data and protect the sensitive data.
SAMPLE_ANALYST	*Fred*	This user is able to unprotect the Name and Amount fields and is able to access the other fields in protected form.

12.2 Data Elements in the Security Policy

The data elements used in the samples are described in the following table.

Table 11-2: List of Data Elements

Data Element	Data Securing Method	Data Element Type	Input Accepted	Description
TOK_NAME	Tokenization	*Alpha-Numeric*	Alphabetic symbols including lowercase (a-z) and uppercase letters (A-Z) and digits from 0 through 9. Min length: 1 Max length: 4080	Data element for the *Customer Name*.
TOK_CREDIT_CARD	Tokenization	*Creditcard*	Digits 0 through 9 with no separators. Min length: 6 Max length: 256	Data element for the *Credit Card* number.
TOK_AMOUNT	Tokenization	*Decimal*	Digits 0 through 9. The sign (+ or -) and decimal point (. or ,) can be used as separators. Min length: 1 Max length: 36	Data element for the *Amount* spent by the customer using the credit card.

Data Element	Data Securing Method	Data Element Type	Input Accepted	Description
TOK_PHONE	Tokenization	*Numeric*	Digits 0 through 9. Min length: 1 Max length: 3933	Data element for the *Phone* number.

12.3 Role-based Permissions for Data Elements in the Sample

The role-based permissions for the data elements used in the samples are described in the following table.

Table 11-3: Role-based Permissions for Data Elements

Roles	TOK_AMOUNT	TOK_NAME	TOK_CREDIT_CARD	TOK_PHONE
SAMPLE_ADMIN	Protect Unprotect	Protect Unprotect	Protect Unprotect	Protect Unprotect
SAMPLE_ANALYST_1	Protect	Protect	Protect	Protect
SAMPLE_ANALYST_2	Unprotect	Unprotect	-	-

12.4 Data Used by the Samples

Table 11-4: Fields and Values in the Sample

ID	Name	Phone	Credit Card	Amount
928724	Hultgren Caylor	9823750987	376235139103947	6959123
928725	Bourne Jose	9823350487	6226600538383292	42964354
928726	Sorce Hatti	9824757883	6226540862865375	7257656
928727	Lorie Garvey	9913730982	5464987835837424	85447788
928728	Belva Beeson	9948752198	5539455602750205	59040774
928729	Hultgren Caylor	9823750987	376235139103947	3245234
928730	Bourne Jose	9823350487	6226600538383292	2300567
928731	Lorie Garvey	9913730982	5464987835837424	85447788
928732	Bourne Jose	9823350487	6226600538383292	3096233
928733	Hultgren Caylor	9823750987	376235139103947	5167763
928734	Lorie Garvey	9913730982	5464987835837424	85447788

12.5 Protecting Data using MapReduce

A MapReduce job in a Hadoop cluster involves sensitive data. You can use the Protegrity MapReduce APIs to protect data when it is saved or retrieved from a protected source.

The Protegrity MapReduce APIs can protect and unprotect the data as defined by the Data security policy.

For more information on the list of available Protegrity MapReduce APIs, refer to section *4.4 MapReduce APIs*.

The following sections describe two sample use cases.

1. A basic use case to demonstrate the functionality of how basic protection and unprotection works using Protegrity MapReduce APIs.
2. A role-based use case to demonstrate the different data access permissions when two users belonging to different roles are viewing the same data.

For ease of illustration, the use cases describe the following two users:

- User with ability to protect the data, thereby accessing the data in protected form.
- User with only access to a few fields from the protected data in cleartext form.

12.5.1 Basic Use Case

This section describes the commands to perform the following functions:

- Display the original data as is.
- Protect the original data.
- Display the data protected using the Protegrity MapReduce API.
- Unprotect the protected data.
- Display the unprotected data.

Ensure that you login as the user *root* before performing the following tasks.

➤ To view the original data:

This command displays the sample data as is.

```
hadoop fs -cat /tmp/basic_sample/sample/basic_sample_data.csv
```

Result: (Original data)

```
ID    ,  NAME           , PHONE        , CREDIT_CARD           , AMOUNT
928724,  Hultgren Caylor, 9823750987   , 376235139103947       , 6959123
928725,  Bourne Jose    , 9823350487   , 6226600538383292      , 42964354
928726,  Sorce Hatti    , 9824757883   , 6226540862865375      , 7257656
928727,  Lorie Garvey   , 9913730982   , 5464987835837424      , 85447788
928728,  Belva Beeson   , 9948752198   , 5539455602750205      , 59040774
928729,  Hultgren Caylor, 9823750987   , 376235139103947       , 3245234
928730,  Bourne Jose    , 9823350487   , 6226600538383292      , 2300567
928731,  Lorie Garvey   , 9913730982   , 5464987835837424      , 85447788
928732,  Bourne Jose    , 9823350487   , 6226600538383292      , 3096233
928733,  Hultgren Caylor, 9823750987   , 376235139103947       , 5167763
928734,  Lorie Garvey   , 9913730982   , 5464987835837424      , 85447788
```

➤ To protect the data:

This command protects the sample data. The data in the Name, Phone, Credit card, and Amount fields is protected.

```
hadoop jar /opt/protegrity/samples/mapreduce/lib/basic*.jar
com.protegrity.samples.mapreduce.ProtectData
/tmp/basic_sample/sample/basic_sample_data.csv
/tmp/basic_sample/protected_mapred_data
```

➤ To view the protected data:

This command displays the protected data.

```
hadoop fs -cat /tmp/basic_sample/protected_mapred_data/part*
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

➤ **To unprotect the data:**

This command unprotects the protected data.

```
hadoop jar /opt/protegrity/samples/mapreduce/lib/basic*.jar
com.protegrity.samples.mapreduce.UnprotectData
/tmp/basic_sample/protected_mapred_data/part*
/tmp/basic_sample/unprotected_mapred_data
```

➤ **To view the unprotected data:**

This command displays the unprotected data.

```
hadoop fs -cat /tmp/basic_sample/unprotected_mapred_data/part*
```

Result: (Unprotected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 9823750987     , 376235139103947      , 6959123
928725,  Bourne Jose     , 9823350487     , 6226600538383292     , 42964354
928726,  Sorce Hatti     , 9824757883     , 6226540862865375     , 7257656
928727,  Lorie Garvey    , 9913730982     , 5464987835837424     , 85447788
928728,  Belva Beeson    , 9948752198     , 5539455602750205     , 59040774
928729,  Hultgren Caylor, 9823750987     , 376235139103947      , 3245234
928730,  Bourne Jose     , 9823350487     , 6226600538383292     , 2300567
928731,  Lorie Garvey    , 9913730982     , 5464987835837424     , 85447788
928732,  Bourne Jose     , 9823350487     , 6226600538383292     , 3096233
928733,  Hultgren Caylor, 9823750987     , 376235139103947      , 5167763
928734,  Lorie Garvey    , 9913730982     , 5464987835837424     , 85447788
```

12.5.2 Role-based Use Cases

This section describes the following two use cases:

- Ingestion User (*John*) ingesting and protecting the data related to credit card transactions.
- Sales Analyst (*Fred*) analyzing credit card transactions to detect the following:
  - Amounts spent by users
  - Fraudulent transactions
  - Repeat users

12.5.2.1 Protect the Credit Card Transactions

John, the Ingestion User, is able to ingest credit card transaction data into HDFS, and protect the sensitive data.

Ensure that you login as the user *John* before performing the following tasks.

➤ **To login with the user John:**

This command logs in the user John.

```
#> su – John
```

Enter the required password for the user *John*, when prompted.

➤ **To view the original data (as John):**

This command displays the sample data as is.

```
hadoop fs -cat /tmp/basic_sample/sample/basic_sample_data.csv
```

Result: (Original data)

```
ID    ,  NAME           , PHONE        , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 9823750987   , 376235139103947      , 6959123
928725,  Bourne Jose    , 9823350487   , 6226600538383292     , 42964354
928726,  Sorce Hatti    , 9824757883   , 6226540862865375     , 7257656
928727,  Lorie Garvey   , 9913730982   , 5464987835837424     , 85447788
928728,  Belva Beeson   , 9948752198   , 5539455602750205     , 59040774
928729,  Hultgren Caylor, 9823750987   , 376235139103947      , 3245234
928730,  Bourne Jose    , 9823350487   , 6226600538383292     , 2300567
928731,  Lorie Garvey   , 9913730982   , 5464987835837424     , 85447788
928732,  Bourne Jose    , 9823350487   , 6226600538383292     , 3096233
928733,  Hultgren Caylor, 9823750987   , 376235139103947      , 5167763
928734,  Lorie Garvey   , 9913730982   , 5464987835837424     , 85447788
```

➤ **To protect the data (as John):**

This command protects the sample data. The sensitive data is protected.

```
hadoop jar /opt/protegrity/samples/mapreduce/lib/basic*.jar
com.protegrity.samples.mapreduce.ProtectData
/tmp/basic_sample/sample/basic_sample_data.csv
/tmp/basic_sample/ingestion_user_protected_mapred_data
```

➤ **To view the protected data (as John):**

This command displays the protected data. The sensitive data appears in tokenized form.

```
hadoop fs -cat /tmp/basic_sample/ingestion_user_protected_mapred_data/part*
```

Result: (Protected data)

```
ID    ,  NAME           , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

➤ **To attempt to unprotect the data (as John):**

This command attempts to unprotect the protected data.

```
hadoop jar /opt/protegrity/samples/mapreduce/lib/basic*.jar
com.protegrity.samples.mapreduce.UnprotectData
/tmp/basic_sample/ingestion_user_protected_mapred_data/part*
/tmp/basic_sample/ingestion_user_unprotected_mapred_data
```

➤ **To attempt to view the unprotected data (as John):**

This command attempts to display the unprotected data. The user *John* will not be able to view the cleartext data as the user does not have permissions to unprotect the data.

```
hadoop fs -cat /tmp/basic_sample/ingestion_user_unprotected_mapred_data/part*
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

➤ **To logout the user John:**

This command logs out the user John.

```
#> exit
```

12.5.2.2 Perform Analysis on Credit Card Transactions

Fred, the Sales Analyst, analyzes the amounts spent by users, the number of fraudulent transactions, and the number of repeat users, using the cleartext data in the Name and Amount fields.

Ensure that you login as the user *Fred* before performing the following tasks.

➤ **To login with the user Fred:**

This command logs in the user Fred.

```
#> su - Fred
```

Enter the required password for the user *Fred*, when prompted.

➤ **To unprotect the data (as Fred):**

This command unprotects the protected data.

```
hadoop jar /opt/protegrity/samples/mapreduce/lib/basic*.jar
com.protegrity.samples.mapreduce.UnprotectData
/tmp/basic_sample/ingestion_user_protected_mapred_data/part*
/tmp/basic_sample/analyst_user_unprotected_mapred_data
```

➤ **To view the unprotected data (as Fred):**

This command displays the unprotected data. The user *Fred* will be able to view the cleartext data for the Name and Amount fields only.

```
hadoop fs -cat /tmp/basic_sample/analyst_user_unprotected_mapred_data/part*
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 27995164409    , 173483871743706      , 6959123
928725,  Bourne Jose     , 87122238232    , 5730496842473502     , 42964354
928726,  Sorce Hatti     , 31934151773    , 6472961686603834     , 7257656
928727,  Lorie Garvey    , 70201301198    , 7277097339102446     , 85447788
928728,  Belva Beeson    , 21190182420    , 3411370995179337     , 59040774
928729,  Hultgren Caylor, 27995164409    , 173483871743706      , 3245234
```

```
928730,  Bourne Jose    , 87122238232    , 5730496842473502     , 2300567
928731,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
928732,  Bourne Jose    , 87122238232    , 5730496842473502     , 3096233
928733,  Hultgren Caylor, 27995164409    , 173483871743706      , 5167763
928734,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
```

Based on the unprotected data that appears, Fred can analyze the following:

- Amounts spent by users.
- Fraudulent credit card transactions – In the sample, the user Lorie Garvey might be a fraudulent user as the transaction amount of 85447788 is repeated thrice across the transactions.
- Number of repeat users – In the sample, there are three repeat users.

The MapReduce sample can be extended to perform actual analysis and isolate the amounts spent, fraudulent transactions, and repeat users.

➤ **To logout the user Fred:**

This command logs out the user Fred.

```
#> exit
```

12.5.3 Sample Code Usage

The MapReduce sample program, described in this section, is an example on how to use the Protegrity MapReduce protector APIs. The sample program utilizes the following two Java classes:

- **ProtectData.java** – This main class calls the Mapper job.
- **ProtectDataMapper.java** – This Mapper class contains the logic to fetch the input data and store the protected content as output.

12.5.3.1 Main Job Class – ProtectData.java

```
package com.protegrity.samples.mapreduce;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class ProtectData extends Configured implements Tool {
       @Override
       public int run(String[] args) throws Exception
       {
    //Create the Job
       Job job = new Job(getConf(), "ProtectData");

    //Set the output key and value class
       job.setOutputKeyClass(NullWritable.class);
       job.setOutputValueClass(Text.class);

    //Set the output key and value class
       job.setMapOutputKeyClass(NullWritable.class);
       job.setMapOutputValueClass(Text.class);
```

```
    //Set the Mapper class which will perform the protect job
       job.setMapperClass(ProtectDataMapper.class);

     //Set number of reducer task
       job.setNumReduceTasks( 0 );

     //Set the input and output Format class
      job.setInputFormatClass(TextInputFormat.class);
       job.setOutputFormatClass(TextOutputFormat.class);

     //Set the jar class
       job.setJarByClass(ProtectData.class);

     //Store the input path and print the input path
       Path input = new Path(args[0]);
       System.out.println(input.getName());
     //Store the output path and print the output path
       Path output = new Path(args[1]);
       System.out.println(output.getName());

     //Add input and set output path
       FileInputFormat.addInputPath(job, new Path(args[0]));
       FileOutputFormat.setOutputPath(job, new Path(args[1]));

     //Call the job
       return job.waitForCompletion(true) ? 0 : 1;
       }

       public static void main(String args[]) throws Exception       {
              System.exit(ToolRunner.run(new Configuration(), new ProtectData(), args));
       }  }
```

12.5.3.2 Mapper Class – ProtectDataMapper.java

```
package com.protegrity.samples.mapreduce;

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

//Need to import the ptyMapReduceProtector class to use the Protegrity MapReduce protector
  import com.protegrity.hadoop.mapreduce.ptyMapReduceProtector;

//Create the Mapper class i.e. ProtectDataMapper which will extends the Mapper Class
  public class ProtectDataMapper extends Mapper<Object, Text, NullWritable, Text> {

//Declare the member variable for the ptyMapReduceProtector class
  private ptyMapReduceProtector mapReduceProtector;

//Declare the Array of Data Elements which will be required to do the protection/unprotection
  private final String[] data_element_names = { "TOK_NAME", "TOK_PHONE", "TOK_CREDIT_CARD",
"TOK_AMOUNT" };

//Initialize the mapreduce protector i.e ptyMapReduceProtector in the default constructor
  public ProtectDataMapper() throws Exception {
        // Create the new object for the class ptyMapReduceProtector
          mapReduceProtector = new ptyMapReduceProtector();

        // Open the session using the method " openSession("0") "
          int openSessionStatus = mapReduceProtector.openSession("0");
}

//Override the map method to parse the text and process it line by line
```

```
//Split the inputs separated by delimiter "," in the line
//Apply the protect/unprotect operation
//Create the output text which will have protected/unprotected outputs separated by delimiter
","
//Write the output text to the context
  @Override
  public void map(Object key, Text value, Context context) throws IOException,
InterruptedException
  {
        // Store the line in a variable strOneLine
        String strOneLine = value.toString();

        // Split the inputs separated by delimiter "," in the line
        StringTokenizer st  = new StringTokenizer(strOneLine, ",");
        // Create the instance of StringBuilder to store the output
        StringBuilder sb = new StringBuilder();

        // Store the no of inputs in a line
        int noOfTokens = st.countTokens();

        if (mapReduceProtector != null)   {
        //Iterate through the string token and apply the protect/unprotect operation
               for (int i = 0; st.hasMoreElements(); i++) {
                      String data = (String)st.nextElement();
                      if(i == 0) {
                            sb.append(new String(data));
                      } else {
//To protect data, call the function protect method with parameters data element and input
data in bytes
//mapReduceProtector.protect( <Data Element> , <Data in bytes> )
//Output will be returned in bytes

//To unprotect data, call the function unprotect method with parameters data element
and input data in bytes
//mapReduceProtector.unprotect( <Data Element> , <Data in bytes> )
//Output will be returned in bytes

                            byte[] bResult = mapReduceProtector.protect(data_element_names[i-
1], data.trim().getBytes());
                       if (bResult != null) {
     // Store the result in string and append it to the output sb
                          sb.append(new String(bResult));
                       }
                       else  {
     // If output will be null, then store the result as "cryptoError" and append it to the
output sb
                          sb.append("cryptoError");
                       }
                      }
                      if(i < noOfTokens -1 )     {
                         // Append delimiter "," at the end of the processed result
                        sb.append(",");
                      }  }  }
        // write the output text to context
        context.write(NullWritable.get(), new Text(sb.toString()));
}

//clean up the session and objects
@Override
protected void finalize() throws Throwable {
        //Close the session
        int closeSessionStatus = mapReduceProtector.closeSession();
        mapReduceProtector = null;
        super.finalize();
        } }
```

12.6 Protecting Data using Hive

You can utilize the Protegrity Hive UDFs to secure data for Hive. The Protegrity Hive UDFs are loaded into Hive during installation. While inserting data to Hive tables, or retrieving data from protected Hive table columns, you can call Protegrity Hive UDFs.

The Protegrity Hive UDFs can protect and unprotect the data as defined by the Data security policy.

For more information on the list of available Protegrity Hive UDFs, refer to section *4.5 Hive UDFs*.

The following sections describe two sample use cases.

1. A basic use case to demonstrate the functionality of how basic protection and unprotection works using Protegrity Hive UDFs.
2. A role-based use case to demonstrate the different data access permissions when two users belonging to different roles are viewing the same data.

For ease of illustration, the use cases describe the following two users:

- User with ability to protect the data, thereby accessing the data in protected form.
- User with only access to a few fields from the protected data in cleartext form.

12.6.1 Basic Use Case

This section describes the commands to perform the following functions:

- Initiate the Hive shell.
- Create a table in Hive using the sample data.
- Define the Protegrity UDFs in Hive.
- Create a Hive table containing the data protected using the Protegrity Hive UDF.
- Display the Hive table which contains the protected data.
- Unprotect the protected data in the Hive table.
- Display the Hive table which contains the unprotected data.

Ensure that you login as the user *root* before performing the following tasks.

➤ **To start the Hive shell:**

This command starts the Hive shell.

```
#> hive
```

➤ **To create *basic_sample* for external table using the *basic_sample_data.csv* file:**

The following commands populate the table named *basic_sample* with the data from the *basic_sample_data.csv* file.

```
#> CREATE TABLE basic_sample (ID STRING, NAME STRING, PHONE STRING, CREDIT_CARD
STRING, AMOUNT STRING) row format delimited fields terminated by ',' stored as
textfile;
#> LOAD DATA LOCAL INPATH '/opt/protegrity/samples/data/basic_sample_data.csv'
OVERWRITE INTO TABLE basic_sample;
```

To define the Protegrity UDFs in Hive:

These commands define the Protegrity UDFs to perform protection and unprotection in Hive.

```
#> create temporary function ptyProtectStr as
'com.protegrity.hive.udf.ptyProtectStr';
#> create temporary function ptyUnprotectStr as
'com.protegrity.hive.udf.ptyUnprotectStr';
```

The UDFs listed here are a subset of the UDFs in the actual product.

For the entire list of UDFs provided by the Protegrity Big Data Protector for Hive, refer to section *4.5 Hive UDFs*.

To create a table with protected data:

This command creates a Hive table with the protected data.

```
#> CREATE TABLE basic_sample_protected as SELECT ID, ptyProtectStr(NAME,
'TOK_NAME') as NAME, ptyProtectStr(PHONE, 'TOK_PHONE') as PHONE,
ptyProtectStr(CREDIT_CARD, 'TOK_CREDIT_CARD') as CREDIT_CARD, ptyProtectStr(AMOUNT,
'TOK_AMOUNT') as AMOUNT FROM basic_sample;
```

To view the protected data table:

This command displays the protected data table. The sensitive data appears in tokenized form.

```
#> SELECT * FROM basic_sample_protected;
```

Result: (Protected data)

ID	NAME	PHONE	CREDIT_CARD	AMOUNT
928724	EnYEwVg3 MOQxQw	27995164409	173483871743706	85924227
928725	4h6NlN FJi9	87122238232	5730496842473502	83764821
928726	Lecwe 48zhNF	31934151773	6472961686603834	49177868
928727	X9lLP BAA8vN	70201301198	7277097339102446	945396991
928728	AYEmh 2CwyvX	21190182420	3411370995179337	976189279
928729	EnYEwVg3 MOQxQw	27995164409	173483871743706	4781777
928730	4h6NlN FJi9	87122238232	5730496842473502	3285956
928731	X9lLP BAA8vN	70201301198	7277097339102446	945396991
928732	4h6NlN FJi9	87122238232	5730496842473502	4112197
928733	EnYEwVg3 MOQxQw	27995164409	173483871743706	63953943
928734	X9lLP BAA8vN	70201301198	7277097339102446	945396991

To unprotect and view the clear data:

This command unprotects and displays the cleartext data. The sensitive data appears in detokenized form.

```
#> SELECT ID, ptyUnprotectStr(NAME, 'TOK_NAME'), ptyUnprotectStr(PHONE,
'TOK_PHONE'), ptyUnprotectStr(CREDIT_CARD, 'TOK_CREDIT_CARD'),
ptyUnprotectStr(AMOUNT, 'TOK_AMOUNT') FROM basic_sample_protected;
```

Result: (Unprotected data)

ID	NAME	PHONE	CREDIT_CARD	AMOUNT
928724	Hultgren Caylor	9823750987	376235139103947	6959123
928725	Bourne Jose	9823350487	6226600538383292	42964354
928726	Sorce Hatti	9824757883	6226540862865375	7257656
928727	Lorie Garvey	9913730982	5464987835837424	85447788
928728	Belva Beeson	9948752198	5539455602750205	59040774
928729	Hultgren Caylor	9823750987	376235139103947	3245234
928730	Bourne Jose	9823350487	6226600538383292	2300567
928731	Lorie Garvey	9913730982	5464987835837424	85447788
928732	Bourne Jose	9823350487	6226600538383292	3096233
928733	Hultgren Caylor	9823750987	376235139103947	5167763
928734	Lorie Garvey	9913730982	5464987835837424	85447788

12.6.2 Role-based Use Cases

This section describes the following two use cases:

- Ingestion User (*John*) ingesting and protecting the data related to credit card transactions.
- Sales Analyst (*Fred*) analyzing credit card transactions to detect the following:
  - Amounts spent by users
  - Fraudulent transactions
  - Repeat users

12.6.2.1 Protect the Credit Card Transactions

John, the Ingestion User, is able to ingest credit card transaction data into HDFS and protect the sensitive data.

Ensure that you login as the user *John* before performing the following tasks.

➤ To login with the user John:

This command logs in the user John.

```
#> su – John
```

Enter the required password for the user *John*, when prompted.

➤ To start the Hive shell (as John):

This command starts the Hive shell.

```
#> hive
```

➤ To create *basic_sample_ingestion_user* for external table using the *basic_sample_data.csv* file (as John):

These commands populate the table named *basic_sample* with the data from the *role_based_sample_data.csv* file.

```
#> CREATE TABLE basic_sample_ingestion_user (ID STRING, NAME STRING, PHONE STRING,
CREDIT_CARD STRING, AMOUNT STRING) row format delimited fields terminated by ','
stored as textfile;
#> LOAD DATA LOCAL INPATH '/opt/protegrity/samples/data/basic_sample_data.csv'
OVERWRITE INTO TABLE basic_sample_ingestion_user;
```

➤ To define the Protegrity UDFs in Hive (as John):

These commands define the Protegrity UDFs to perform protection and unprotection in Hive.

```
#> create temporary function ptyProtectStr as
'com.protegrity.hive.udf.ptyProtectStr';
#> create temporary function ptyUnprotectStr as
'com.protegrity.hive.udf.ptyUnprotectStr';
```

➤ To create a table with protected data (as John):

This command creates a table with the protected data.

```
#> CREATE TABLE basic_sample_protected_ingestion_user as SELECT ID,
ptyProtectStr(NAME, 'TOK_NAME') as NAME, ptyProtectStr(PHONE, 'TOK_PHONE') as
PHONE, ptyProtectStr(CREDIT_CARD, 'TOK_CREDIT_CARD') as CREDIT_CARD,
ptyProtectStr(AMOUNT, 'TOK_AMOUNT') as AMOUNT FROM basic_sample_ingestion_user;
```

➤ To view the protected data (as John):

This command displays the protected data.

```
#> SELECT * FROM basic_sample_protected_ingestion_user;
```

Result: (Protected data)

ID	NAME	PHONE	CREDIT_CARD	AMOUNT
928724	EnYEwVg3 MOQxQw	27995164409	173483871743706	85924227
928725	4h6NlN FJi9	87122238232	5730496842473502	83764821
928726	Lecwe 48zhNF	31934151773	6472961686603834	49177868
928727	X9lLP BAA8vN	70201301198	7277097339102446	945396991
928728	AYEmh 2CwyvX	21190182420	3411370995179337	976189279
928729	EnYEwVg3 MOQxQw	27995164409	173483871743706	4781777
928730	4h6NlN FJi9	87122238232	5730496842473502	3285956
928731	X9lLP BAA8vN	70201301198	7277097339102446	945396991
928732	4h6NlN FJi9	87122238232	5730496842473502	4112197
928733	EnYEwVg3 MOQxQw	27995164409	173483871743706	63953943
928734	X9lLP BAA8vN	70201301198	7277097339102446	945396991

➤ **To attempt to unprotect and view the clear data (as John):**

This command attempts to unprotect and display the unprotected data. The user *John* will not be able to view the cleartext data as the user does not have permissions to unprotect the data.

```
#> SELECT ID, ptyUnprotectStr(NAME, 'TOK_NAME'), ptyUnprotectStr(PHONE,
'TOK_PHONE'), ptyUnprotectStr(CREDIT_CARD, 'TOK_CREDIT_CARD'),
ptyUnprotectStr(AMOUNT, 'TOK_AMOUNT') FROM basic_sample_protected_ingestion_user;
```

Result: (Protected data)

ID	NAME	PHONE	CREDIT_CARD	AMOUNT
928724	EnYEwVg3 MOQxQw	27995164409	173483871743706	85924227
928725	4h6NlN FJi9	87122238232	5730496842473502	83764821
928726	Lecwe 48zhNF	31934151773	6472961686603834	49177868
928727	X9lLP BAA8vN	70201301198	7277097339102446	945396991
928728	AYEmh 2CwyvX	21190182420	3411370995179337	976189279
928729	EnYEwVg3 MOQxQw	27995164409	173483871743706	4781777
928730	4h6NlN FJi9	87122238232	5730496842473502	3285956
928731	X9lLP BAA8vN	70201301198	7277097339102446	945396991
928732	4h6NlN FJi9	87122238232	5730496842473502	4112197
928733	EnYEwVg3 MOQxQw	27995164409	173483871743706	63953943
928734	X9lLP BAA8vN	70201301198	7277097339102446	945396991

➤ **To logout the user John:**

This command logs out the user John.

```
#> exit
```

12.6.2.2 Perform Analysis on Credit Card Transactions

Fred, the Sales Analyst, analyzes the amounts spent by users, the number of fraudulent transactions, and the number of repeat users, using the cleartext data in the Name and Amount fields.

Ensure that you login as the user *Fred* before performing the following tasks.

➤ **To login with the user Fred:**

This command logs in the user Fred.

```
#> su – Fred
```

Enter the required password for the user *Fred*, when prompted.

➤ **To start the Hive shell (as Fred):**

This command starts the Hive shell.

```
#> hive
```

➤ **To define the Protegrity UDFs in Hive (as Fred):**

The following commands define the Protegrity UDFs to perform protection and unprotection in Hive.

```
#> create temporary function ptyProtectStr as
'com.protegrity.hive.udf.ptyProtectStr';
#> create temporary function ptyUnprotectStr as
'com.protegrity.hive.udf.ptyUnprotectStr';
```

➤ **To unprotect and view the clear data (as Fred):**

This command unprotects the protected data and displays the unprotected data. The user *Fred* will be able to view the cleartext data for the Name and Amount fields only.

```
#> SELECT ID, ptyUnprotectStr(NAME, 'TOK_NAME'), ptyUnprotectStr(PHONE,
'TOK_PHONE'), ptyUnprotectStr(CREDIT_CARD, 'TOK_CREDIT_CARD'),
ptyUnprotectStr(AMOUNT, 'TOK_AMOUNT') FROM basic_sample_protected_ingestion_user;
```

Result: (Protected data)

ID	NAME	PHONE	CREDIT_CARD	AMOUNT
928724	Hultgren Caylor	27995164409	173483871743706	6959123
928725	Bourne Jose	87122238232	5730496842473502	42964354
928726	Sorce Hatti	31934151773	6472961686603834	7257656
928727	Lorie Garvey	70201301198	7277097339102446	85447788
928728	Belva Beeson	21190182420	3411370995179337	59040774
928729	Hultgren Caylor	27995164409	173483871743706	3245234
928730	Bourne Jose	87122238232	5730496842473502	2300567
928731	Lorie Garvey	70201301198	7277097339102446	85447788
928732	Bourne Jose	87122238232	5730496842473502	3096233
928733	Hultgren Caylor	27995164409	173483871743706	5167763
928734	Lorie Garvey	70201301198	7277097339102446	85447788

Based on the unprotected data that appears, Fred can analyze the following:

- Amounts spent by users
- Fraudulent credit card transactions – In the sample, the user Lorie Garvey might be a fraudulent user as the transaction amount of 85447788 is repeated thrice across the transactions.
- Number of repeat users – In the sample, there are three repeat users.

The Hive sample can be extended to perform actual analysis and isolate the amounts spent, fraudulent transactions, and repeat users.

➤ **To logout the user Fred:**

This command logs out the user Fred.

```
#> exit
```

12.7 Protecting Data using Pig

You can utilize the Protegrity Pig UDFs to secure data, while running Pig jobs. While inserting or retrieving data using Pig, you can call Protegrity Pig UDFs.

The Protegrity Pig UDFs can protect and unprotect the data as defined by the Data security policy.

For more information on the list of available Protegrity Pig UDFs, refer to section *4.6 Pig UDFs*.

The following sections describe two sample use cases.

1. A basic use case to demonstrate the functionality of how basic protection and unprotection works using Protegrity Pig UDFs.
2. A role-based use case to demonstrate the different data access permissions when two users belonging to different roles are viewing the same data.

For ease of illustration, the use cases describe the following two users:

- User with ability to protect the data, thereby accessing the data in protected form.
- User with only access to a few fields from the protected data in cleartext form.

12.7.1 Basic Use Case

This section describes the commands to perform the following functions:

- Initiate the Pig shell.
- Define the Protegrity UDFs in Pig.
- Define a variable in Pig using the sample data.
- Display the original data as is.
- Protect the data in the defined variable using the Protegrity Pig UDF.
- Display the protected data.
- Unprotect the protected data.
- Display the unprotected data.

Ensure that you login as the user *root* before performing the following tasks.

➤ To start the Pig shell:

This command starts the Pig shell.

```
#> pig
```

➤ To define the UDFs:

These commands define the Protegrity UDFs to perform protection and unprotection in Pig.

```
grunt> DEFINE ptyProtectStr com.protegrity.pig.udf.ptyProtectStr;
grunt> DEFINE ptyUnprotectStr com.protegrity.pig.udf.ptyUnprotectStr;
```

➤ To create a variable with the original data:

This command creates a variable with the sample data.

```
grunt> basic_sample = LOAD '/tmp/basic_sample/sample/basic_sample_data.csv' using
PigStorage(',') AS (ID:chararray, NAME:chararray, PHONE:chararray,
CREDIT_CARD:chararray, AMOUNT:chararray);
```

➤ To view the original data:

This command displays the sample data.

```
grunt> dump basic_sample;
```

➤ To protect the data:

This command protects the data.

```
grunt> basic_sample_protected = FOREACH basic_sample GENERATE ID,
ptyProtectStr(NAME, 'TOK_NAME') as NAME:chararray, ptyProtectStr(PHONE,
'TOK_PHONE') as PHONE:chararray, ptyProtectStr(CREDIT_CARD, 'TOK_CREDIT_CARD') as
CREDIT_CARD:chararray, ptyProtectStr(AMOUNT, 'TOK_AMOUNT') as AMOUNT:chararray;
```

➤ To view the protected data:

This command displays the protected data.

```
grunt> dump basic_sample_protected;
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

➤ **To unprotect the data:**

This command unprotects the protected data.

```
grunt> basic_sample_unprotected = FOREACH basic_sample_protected GENERATE ID,
ptyUnprotectStr(NAME, 'TOK_NAME') as NAME:chararray, ptyUnprotectStr(PHONE,
'TOK_PHONE') as PHONE:chararray, ptyUnprotectStr(CREDIT_CARD, 'TOK_CREDIT_CARD') as
CREDIT_CARD:chararray, ptyUnprotectStr(AMOUNT, 'TOK_AMOUNT') as AMOUNT:chararray;
```

➤ **To view the unprotected data:**

This command displays the unprotected data.

```
grunt> dump basic_sample_unprotected;
```

Result: (Unprotected data)

```
(ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT)
(928724,  Hultgren Caylor, 9823750987    , 376235139103947      , 6959123)
(928725,  Bourne Jose    , 9823350487    , 6226600538383292     , 42964354)
(928726,  Sorce Hatti    , 9824757883    , 6226540862865375     , 7257656)
(928727,  Lorie Garvey   , 9913730982    , 5464987835837424     , 85447788)
(928728,  Belva Beeson   , 9948752198    , 5539455602750205     , 59040774)
(928729,  Hultgren Caylor, 9823750987    , 376235139103947      , 3245234)
(928730,  Bourne Jose    , 9823350487    , 6226600538383292     , 2300567)
(928731,  Lorie Garvey   , 9913730982    , 5464987835837424     , 85447788)
(928732,  Bourne Jose    , 9823350487    , 6226600538383292     , 3096233)
(928733,  Hultgren Caylor, 9823750987    , 376235139103947      , 5167763)
(928734,  Lorie Garvey   , 9913730982    , 5464987835837424     , 85447788)
```

12.7.2 Role-based Use Cases

This section describes the following two use cases:

- Ingestion User (*John*) ingesting and protecting the data related to credit card transactions.
- Sales Analyst (*Fred*) analyzing credit card transactions to detect the following:
  - Amounts spent by users
  - Fraudulent transactions
  - Repeat users

12.7.2.1 Protect the Credit Card Transactions

John, the Ingestion User, is able to ingest credit card transaction data into HDFS and protect the sensitive data.

Ensure that you login as the user *John* before performing the following tasks.

➤ **To login with the user John:**

This command logs in the user John.

```
#> su – John
```

Enter the required password for the user *John*, when prompted.

➤ **To start the Pig shell (as John):**

This command starts the Pig shell.

```
#> pig
```

➤ **To define the UDFs (as John):**

These commands define the Protegrity UDFs to perform protection and unprotection in Pig.

```
grunt> DEFINE ptyProtectStr com.protegrity.pig.udf.ptyProtectStr;
grunt> DEFINE ptyUnprotectStr com.protegrity.pig.udf.ptyUnprotectStr;
```

➤ **To create a variable with the original data (as John):**

This command creates a variable with the sample data.

```
grunt> basic_sample = LOAD '/tmp/basic_sample/sample/basic_sample_data.csv' using
PigStorage(',') AS (ID:chararray, NAME:chararray, PHONE:chararray,
CREDIT_CARD:chararray, AMOUNT:chararray);
```

➤ **To view the original data (as John):**

This command displays the sample data.

```
grunt> dump basic_sample;
```

Result: (Original data)

```
ID    ,  NAME           , PHONE        , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 9823750987   , 376235139103947      , 6959123
928725,  Bourne Jose    , 9823350487   , 6226600538383292     , 42964354
928726,  Sorce Hatti    , 9824757883   , 6226540862865375     , 7257656
928727,  Lorie Garvey   , 9913730982   , 5464987835837424     , 85447788
928728,  Belva Beeson   , 9948752198   , 5539455602750205     , 59040774
928729,  Hultgren Caylor, 9823750987   , 376235139103947      , 3245234
928730,  Bourne Jose    , 9823350487   , 6226600538383292     , 2300567
928731,  Lorie Garvey   , 9913730982   , 5464987835837424     , 85447788
928732,  Bourne Jose    , 9823350487   , 6226600538383292     , 3096233
928733,  Hultgren Caylor, 9823750987   , 376235139103947      , 5167763
928734,  Lorie Garvey   , 9913730982   , 5464987835837424     , 85447788
```

➤ **To protect the data (as John):**

This command protects the sample data.

```
grunt> basic_sample_protected = FOREACH basic_sample GENERATE ID,
ptyProtectStr(NAME, 'TOK_NAME') as NAME:chararray, ptyProtectStr(PHONE,
'TOK_PHONE') as PHONE:chararray, ptyProtectStr(CREDIT_CARD, 'TOK_CREDIT_CARD') as
CREDIT_CARD:chararray, ptyProtectStr(AMOUNT, 'TOK_AMOUNT') as AMOUNT:chararray;
```

To view the protected data (as John):

This command displays the protected data.

```
grunt> dump basic_sample_protected;
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

To store the protected data (as John):

This command saves the protected data.

```
grunt> STORE basic_sample_protected INTO '/tmp/basic_sample/basic_sample_protected'
using PigStorage(',');
```

To attempt to unprotect the data (as John):

This command attempts to unprotect the protected data.

```
grunt> basic_sample_unprotected = FOREACH basic_sample_protected GENERATE ID,
ptyUnprotectStr(NAME, 'TOK_NAME') as NAME:chararray, ptyUnprotectStr(PHONE,
'TOK_PHONE') as PHONE:chararray, ptyUnprotectStr(CREDIT_CARD, 'TOK_CREDIT_CARD') as
CREDIT_CARD:chararray, ptyUnprotectStr(AMOUNT, 'TOK_AMOUNT') as AMOUNT:chararray;
```

To attempt to view the unprotected data (as John):

This command attempts to displays the unprotected data. The user *John* will not be able to view the cleartext data as the user does not have permissions to unprotect the data.

```
grunt> dump basic_sample_unprotected;
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

To logout the user John:

This command logs out the user John.

```
#> exit
```

12.7.2.2 Perform Analysis on Credit Card Transactions

Fred, the Sales Analyst, analyzes the amounts spent by users, the number of fraudulent transactions, and the number of repeat users, using the cleartext data in the Name and Amount fields.

Ensure that you login as the user *Fred* before performing the following tasks.

To login with the user Fred:

This command logs in the user Fred.

```
#> su – Fred
```

Enter the required password for the user *Fred*, when prompted.

To start the Pig shell (as Fred):

This command starts the Pig shell.

```
#> pig
```

To define the Protegrity UDF's in Pig (as Fred):

These commands define the Protegrity UDFs to perform protection and unprotection in Pig.

```
grunt> DEFINE ptyProtectStr com.protegrity.pig.udf.ptyProtectStr;
grunt> DEFINE ptyUnprotectStr com.protegrity.pig.udf.ptyUnprotectStr;
```

To unprotect the data (as Fred):

The following commands unprotect the protected data.

```
grunt> basic_sample_protected = LOAD '/tmp/basic_sample/basic_sample_protected' using
PigStorage(',') AS (ID:chararray, NAME:chararray, PHONE:chararray,
CREDIT_CARD:chararray, AMOUNT:chararray);
grunt> basic_sample_unprotected = FOREACH basic_sample_protected GENERATE ID,
ptyUnprotectStr(NAME, 'TOK_NAME') as NAME:chararray, ptyUnprotectStr(PHONE, 'TOK_PHONE')
as PHONE:chararray, ptyUnprotectStr(CREDIT_CARD, 'TOK_CREDIT_CARD') as
CREDIT_CARD:chararray, ptyUnprotectStr(AMOUNT, 'TOK_AMOUNT') as AMOUNT:chararray;
```

To view the unprotected data (as Fred):

This command displays the unprotected data. The user Fred will be able to view the cleartext data for the Name and Amount fields only.

```
grunt> dump basic_sample_unprotected;
```

Result: (Protected data)

```
ID    ,  NAME           , PHONE          , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 27995164409    , 173483871743706      , 6959123
928725,  Bourne Jose    , 87122238232    , 5730496842473502     , 42964354
928726,  Sorce Hatti    , 31934151773    , 6472961686603834     , 7257656
928727,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
928728,  Belva Beeson   , 21190182420    , 3411370995179337     , 59040774
928729,  Hultgren Caylor, 27995164409    , 173483871743706      , 3245234
928730,  Bourne Jose    , 87122238232    , 5730496842473502     , 2300567
928731,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
928732,  Bourne Jose    , 87122238232    , 5730496842473502     , 3096233
928733,  Hultgren Caylor, 27995164409    , 173483871743706      , 5167763
928734,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
```

Based on the unprotected data that appears, Fred can analyze the following:

- Amounts spent by users
- Fraudulent credit card transactions – In the sample, the user Lorie Garvey might be a fraudulent user as the transaction amount of 85447788 is repeated thrice across the transactions.
- Number of repeat users – In the sample, there are three repeat users.

The Pig sample can be extended to perform actual analysis and isolate the amounts spent, fraudulent transactions, and repeat users.

➤ To logout the user Fred:

This command exits the user Fred.

```
#> exit
```

12.8 Protecting Data using HBase

The data which is stored in each row of an HBase table can be protected through tokenization. Based on the access privileges of the users, the data stored in each row of the HBase table, can be made available to the users in cleartext form or tokenized form.

The following sections describe two sample use cases.

1. A basic use case to demonstrate the functionality of how basic protection and unprotection works using HBase.
2. A role-based use case to demonstrate the different data access permissions when two users belonging to different roles are viewing the same data.

For ease of illustration, the use cases describe the following two users:

- User with ability to protect the data, thereby accessing the data in protected form.
- User with only access to a few fields from the protected data in cleartext form.

12.8.1 Basic Use Case

This section describes the commands to perform the following functions:

- Create an HBase table.
- Insert the sample data in one row of the HBase table.
- Display the protected data from the row in the HBase table.
- Display the cleartext data from the row in the HBase table.
- Exit the HBase shell.

Ensure that you login as the user *root* before performing the following tasks.

➤ To create a table:

This command creates a table from the sample data.

```
hbase#> create 'basic_sample', {
NAME => 'column_family', METADATA => {
'DATA_ELEMENT:NAME'=>'TOK_NAME',
'DATA_ELEMENT:PHONE'=>'TOK_PHONE',
'DATA_ELEMENT:CREDIT_CARD'=>'TOK_CREDIT_CARD',
'DATA_ELEMENT:AMOUNT'=>'TOK_AMOUNT'}}
```

➤ To insert the data in one row:

This command inserts data in one row. The data inserted in the row is protected transparently.

```
hbase#> put 'basic_sample','928724', 'column_family:NAME', 'Hultgren Caylor'
hbase#> put 'basic_sample','928724', 'column_family:PHONE', '9823750987'
hbase#> put 'basic_sample','928724', 'column_family:CREDIT_CARD', '376235139103947'
hbase#> put 'basic_sample','928724', 'column_family:AMOUNT', '6959123'
```

➤ **To view the protected data:**

This command displays the protected data.

```
hbase#> scan 'basic_sample', { ATTRIBUTES => {'BYPASS_COPROCESSOR'=>'1'}}
```

Result: (Protected data)

```
ROW         COLUMN+CELL
928724      column=column_family:AMOUNT, timestamp=1415647620729, value=85924227
928724      column=column_family:CREDIT_CARD, timestamp=1415647617993, value=173483871743706
928724      column=column_family:NAME, timestamp=1415647617914, value=EnYEwVg3 MOQxQw
928724      column=column_family:PHONE, timestamp=1415647617971, value=27995164409
```

➤ **To view the unprotected data:**

This command displays the unprotected data.

```
hbase#> scan 'basic_sample'
```

Result: (Unprotected data)

```
ROW         COLUMN+CELL
928724      column=column_family:CREDIT_CARD, timestamp=1413848942769, value=376235139103947
928724      column=column_family:NAME, timestamp=1413848942769, value=Hultgren Caylor
928724      column=column_family:PHONE, timestamp=1413848942769, value=9823750987
928724      column=column_family:AMOUNT, timestamp=1413848942769, value=6959123
```

➤ **To exit the HBase shell:**

This command exits the HBase shell.

```
hbase#> exit
```

12.8.2 Role-based Use Cases

This section describes the following two use cases:

- Ingestion User (*John*) ingesting and protecting the data related to credit card transactions.
- Sales Analyst (*Fred*) analyzing credit card transactions to detect the following:
  - Amounts spent by users
  - Fraudulent transactions
  - Repeat users

12.8.2.1 Protect the Credit Card Transactions

John, the Ingestion User, is able to ingest credit card transaction data into HDFS and protect the sensitive data.

Ensure that you login as the user *John* before performing the following tasks.

➤ **To login with the user John:**

This command logs in the user John.

```
#> su - John
```

Enter the required password for the user *John*, when prompted.

➤ **To start the HBase shell (as John):**

This command starts the HBase shell.

```
#> hbase shell
```

➤ **To create the table (as John):**

This command creates the table.

```
hbase#> create 'basic_sample_ingestion_user', {
NAME => 'column_family', METADATA => {
'DATA_ELEMENT:NAME'=>'TOK_NAME',
'DATA_ELEMENT:PHONE'=>'TOK_PHONE',
'DATA_ELEMENT:CREDIT_CARD'=>'TOK_CREDIT_CARD',
'DATA_ELEMENT:AMOUNT'=>'TOK_AMOUNT'}}
```

➤ **To insert the data in one row (as John):**

This command inserts the data in one row. The data inserted in the row is protected transparently.

```
hbase#> put 'basic_sample_ingestion_user','928724', 'column_family:NAME', 'Hultgren
Caylor'
hbase#> put 'basic_sample_ingestion_user','928724', 'column_family:PHONE', '9823750987'
hbase#> put 'basic_sample_ingestion_user','928724', 'column_family:CREDIT_CARD',
'376235139103947'
hbase#> put 'basic_sample_ingestion_user','928724', 'column_family:AMOUNT', '6959123'
```

To insert data in the other rows of the HBase table, execute the following commands.

```
hbase#> put 'basic_sample_ingestion_user','928725', 'column_family:NAME', 'Bourne Jose'
hbase#> put 'basic_sample_ingestion_user','928725', 'column_family:PHONE', '9823350487'
hbase#> put 'basic_sample_ingestion_user','928725', 'column_family:CREDIT_CARD',
'6226600538383292'
hbase#> put 'basic_sample_ingestion_user','928725', 'column_family:AMOUNT', '42964352'
hbase#> put 'basic_sample_ingestion_user','928726', 'column_family:NAME', 'Sorce Hatti'
hbase#> put 'basic_sample_ingestion_user','928726', 'column_family:PHONE', '9824757883'
hbase#> put 'basic_sample_ingestion_user','928726', 'column_family:CREDIT_CARD',
'6226540862865375'
hbase#> put 'basic_sample_ingestion_user','928726', 'column_family:AMOUNT', '7257656'
hbase#> put 'basic_sample_ingestion_user','928727', 'column_family:NAME', 'Lorie Garvey'
hbase#> put 'basic_sample_ingestion_user','928727', 'column_family:PHONE', '9913730982'
hbase#> put 'basic_sample_ingestion_user','928727', 'column_family:CREDIT_CARD',
'5464987835837424'
hbase#> put 'basic_sample_ingestion_user','928727', 'column_family:AMOUNT', '85447788'
hbase#> put 'basic_sample_ingestion_user','928728', 'column_family:NAME', 'Belva Beeson'
hbase#> put 'basic_sample_ingestion_user','928728', 'column_family:PHONE', '9948752198'
hbase#> put 'basic_sample_ingestion_user','928728', 'column_family:CREDIT_CARD',
'5539455602750205'
hbase#> put 'basic_sample_ingestion_user','928728', 'column_family:AMOUNT', '59040774'
hbase#> put 'basic_sample_ingestion_user','928729', 'column_family:NAME', 'Hultgren
Caylor'
hbase#> put 'basic_sample_ingestion_user','928729', 'column_family:PHONE', '9823750987'
hbase#> put 'basic_sample_ingestion_user','928729', 'column_family:CREDIT_CARD',
'376235139103947'
hbase#> put 'basic_sample_ingestion_user','928729', 'column_family:AMOUNT', '3245234'
hbase#> put 'basic_sample_ingestion_user','928730', 'column_family:NAME', 'Lorie Garvey'
hbase#> put 'basic_sample_ingestion_user','928730', 'column_family:PHONE', '9913730982'
hbase#> put 'basic_sample_ingestion_user','928730', 'column_family:CREDIT_CARD',
'5464987835837424'
hbase#> put 'basic_sample_ingestion_user','928730', 'column_family:AMOUNT', '85447788'
hbase#> put 'basic_sample_ingestion_user','928731', 'column_family:NAME', 'Bourne Jose'
hbase#> put 'basic_sample_ingestion_user','928731', 'column_family:PHONE', '9823350487'
hbase#> put 'basic_sample_ingestion_user','928731', 'column_family:CREDIT_CARD',
'6226600538383292'
```

```
hbase#> put 'basic_sample_ingestion_user','928731', 'column_family:AMOUNT', '2300567'
hbase#> put 'basic_sample_ingestion_user','928732', 'column_family:NAME', 'Lorie Garvey'
hbase#> put 'basic_sample_ingestion_user','928732', 'column_family:PHONE', '9913730982'
hbase#> put 'basic_sample_ingestion_user','928732', 'column_family:CREDIT_CARD',
'5464987835837424'
hbase#> put 'basic_sample_ingestion_user','928732', 'column_family:AMOUNT', '85447788'
```

➤ **To view the protected data (as John):**

This command displays the protected data.

```
hbase#> scan 'basic_sample_ingestion_user', { ATTRIBUTES =>
{'BYPASS_COPROCESSOR'=>'1'}}
```

Result: (Protected data)

```
ROW         COLUMN+CELL
928724      column=column_family:AMOUNT, timestamp=1415685268300, value=07329754
928724      column=column_family:CREDIT_CARD, timestamp=1415685268257, value=173483871743706
928724      column=column_family:NAME, timestamp=1415685268073, value=EnYEwVg3 MOQxQw
928724      column=column_family:PHONE, timestamp=1415685268209, value=27995164409
928725      column=column_family:AMOUNT, timestamp=1415685268475, value=772750955
928725      column=column_family:CREDIT_CARD, timestamp=1415685268435, value=5730496842473502
928725      column=column_family:NAME, timestamp=1415685268338, value=4h6NlN FJi9
928725      column=column_family:PHONE, timestamp=1415685268387, value=87122238232
928726      column=column_family:AMOUNT, timestamp=1415685268778, value=12551106
928726      column=column_family:CREDIT_CARD, timestamp=1415685268743, value=6472961686603834
928726      column=column_family:NAME, timestamp=1415685268527, value=Lecwe 48zhNF
928726      column=column_family:PHONE, timestamp=1415685268596, value=31934151773
928727      column=column_family:AMOUNT, timestamp=1415685269001, value=768063717
928727      column=column_family:CREDIT_CARD, timestamp=1415685268943, value=7277097339102446
928727      column=column_family:NAME, timestamp=1415685268830, value=X9lLP BAA8vN
928727      column=column_family:PHONE, timestamp=1415685268898, value=70201301198
928728      column=column_family:AMOUNT, timestamp=1415685269195, value=943884472
928728      column=column_family:CREDIT_CARD, timestamp=1415685269153, value=3411370995179337
928728      column=column_family:NAME, timestamp=1415685269049, value=AYEmh 2CwyvX
928728      column=column_family:PHONE, timestamp=1415685269116, value=21190182420
928729      column=column_family:AMOUNT, timestamp=1415685269353, value=7918658
928729      column=column_family:CREDIT_CARD, timestamp=1415685269315, value=173483871743706
928729      column=column_family:NAME, timestamp=1415685269235, value=EnYEwVg3 MOQxQw
928729      column=column_family:PHONE, timestamp=1415685269279, value=27995164409
928730      column=column_family:AMOUNT, timestamp=1415685269499, value=768063717
928730      column=column_family:CREDIT_CARD, timestamp=1415685269464, value=7277097339102446
928730      column=column_family:NAME, timestamp=1415685269399, value=X9lLP BAA8vN
928730      column=column_family:PHONE, timestamp=1415685269431, value=70201301198
928731      column=column_family:AMOUNT, timestamp=1415685269645, value=9617663
928731      column=column_family:CREDIT_CARD, timestamp=1415685269613, value=5730496842473502
928731      column=column_family:NAME, timestamp=1415685269541, value=4h6NlN FJi9
928731      column=column_family:PHONE, timestamp=1415685269580, value=87122238232
928732      column=column_family:AMOUNT, timestamp=1415685271454, value=768063717
928732      column=column_family:CREDIT_CARD, timestamp=1415685269748, value=7277097339102446
928732      column=column_family:NAME, timestamp=1415685269684, value=X9lLP BAA8vN
928732      column=column_family:PHONE, timestamp=1415685269716, value=70201301198
```

➤ **To attempt to view the unprotected data (as John):**

This command attempts to display the unprotected data. The user *John* will not be able to view the cleartext data as the user does not have permissions to unprotect the data.

```
hbase#> scan 'basic_sample_ingestion_user'
```

Result: (Protected data)

```
ROW         COLUMN+CELL
928724      column=column_family:AMOUNT, timestamp=1415685268300, value=07329754
928724      column=column_family:CREDIT_CARD, timestamp=1415685268257, value=173483871743706
928724      column=column_family:NAME, timestamp=1415685268073, value=EnYEwVg3 MOQxQw
928724      column=column_family:PHONE, timestamp=1415685268209, value=27995164409
928725      column=column_family:AMOUNT, timestamp=1415685268475, value=772750955
928725      column=column_family:CREDIT_CARD, timestamp=1415685268435, value=5730496842473502
928725      column=column_family:NAME, timestamp=1415685268338, value=4h6NlN FJi9
928725      column=column_family:PHONE, timestamp=1415685268387, value=87122238232
928726      column=column_family:AMOUNT, timestamp=1415685268778, value=12551106
928726      column=column_family:CREDIT_CARD, timestamp=1415685268743, value=6472961686603834
928726      column=column_family:NAME, timestamp=1415685268527, value=Lecwe 48zhNF
928726      column=column_family:PHONE, timestamp=1415685268596, value=31934151773
928727      column=column_family:AMOUNT, timestamp=1415685269001, value=768063717
928727      column=column_family:CREDIT_CARD, timestamp=1415685268943, value=7277097339102446
928727      column=column_family:NAME, timestamp=1415685268830, value=X9lLP BAA8vN
928727      column=column_family:PHONE, timestamp=1415685268898, value=70201301198
928728      column=column_family:AMOUNT, timestamp=1415685269195, value=943884472
928728      column=column_family:CREDIT_CARD, timestamp=1415685269153, value=3411370995179337
928728      column=column_family:NAME, timestamp=1415685269049, value=AYEmh 2CwyvX
928728      column=column_family:PHONE, timestamp=1415685269116, value=21190182420
928729      column=column_family:AMOUNT, timestamp=1415685269353, value=7918658
928729      column=column_family:CREDIT_CARD, timestamp=1415685269315, value=173483871743706
928729      column=column_family:NAME, timestamp=1415685269235, value=EnYEwVg3 MOQxQw
928729      column=column_family:PHONE, timestamp=1415685269279, value=27995164409
928730      column=column_family:AMOUNT, timestamp=1415685269499, value=768063717
928730      column=column_family:CREDIT_CARD, timestamp=1415685269464, value=7277097339102446
928730      column=column_family:NAME, timestamp=1415685269399, value=X9lLP BAA8vN
928730      column=column_family:PHONE, timestamp=1415685269431, value=70201301198
928731      column=column_family:AMOUNT, timestamp=1415685269645, value=9617663
928731      column=column_family:CREDIT_CARD, timestamp=1415685269613, value=5730496842473502
928731      column=column_family:NAME, timestamp=1415685269541, value=4h6NlN FJi9
928731      column=column_family:PHONE, timestamp=1415685269580, value=87122238232
928732      column=column_family:AMOUNT, timestamp=1415685271454, value=768063717
928732      column=column_family:CREDIT_CARD, timestamp=1415685269748, value=7277097339102446
928732      column=column_family:NAME, timestamp=1415685269684, value=X9lLP BAA8vN
928732      column=column_family:PHONE, timestamp=1415685269716, value=70201301198
```

➤ **To exit the HBase shell (as John):**

This command exits the HBase shell.

```
hbase#> exit
```

➤ **To logout the user John:**

This command logs out the user John.

```
#> exit
```

12.8.2.2 Perform Analysis on Credit Card Transactions

Fred, the Sales Analyst, analyzes the amounts spent by users, the number of fraudulent transactions, and the number of repeat users, using the cleartext data in the Name and Amount fields.

Ensure that you login as the user *Fred* before performing the following tasks.

➤ **To login with the user Fred:**

This command logs in the user Fred.

```
#> su – Fred
```

Enter the required password for the user *Fred*, when prompted.

➤ **To start the HBase shell (as Fred):**

This command starts the HBase shell.

```
#> hbase shell
```

➤ **To view the unprotected data (as Fred):**

This command displays the unprotected data. The user *Fred* will be able to view the cleartext data for the Name and Amount fields only.

```
hbase#> scan 'basic_sample_ingestion_user'
```

Result: (Unprotected data)

```
ROW       COLUMN+CELL
928724    column=column_family:AMOUNT, timestamp=223372036854775807, value=6959123
928724    column=column_family:CREDIT_CARD, timestamp=223372036854775807, value=173483871743706
928724    column=column_family:NAME, timestamp=223372036854775807, value=Hultgren Caylor
928724    column=column_family:PHONE, timestamp=223372036854775807, value=27995164409
928725    column=column_family:AMOUNT, timestamp=9223372036854775807, value=42964352
928725    column=column_family:CREDIT_CARD, timestamp=223372036854775807, value=5730496842473502
928725    column=column_family:NAME, timestamp=223372036854775807, value=Bourne Jose
928725    column=column_family:PHONE, timestamp=223372036854775807, value=87122238232
928726    column=column_family:AMOUNT, timestamp=9223372036854775807, value=7257656
928726    column=column_family:CREDIT_CARD, timestamp=9223372036854775807, value=6472961686603834
928726    column=column_family:NAME, timestamp=9223372036854775807, value=Sorce Hatti
928726    column=column_family:PHONE, timestamp=9223372036854775807, value=31934151773
928727    column=column_family:AMOUNT, timestamp=9223372036854775807, value=85447788
928727    column=column_family:CREDIT_CARD, timestamp=9223372036854775807, value=7277097339102446
928727    column=column_family:NAME, timestamp=9223372036854775807, value=Lorie Garvey
928727    column=column_family:PHONE, timestamp=9223372036854775807, value=70201301198
928728    column=column_family:AMOUNT, timestamp=9223372036854775807, value=59040774
928728    column=column_family:CREDIT_CARD, timestamp=9223372036854775807, value=3411370995179337
928728    column=column_family:NAME, timestamp=9223372036854775807, value=Belva Beeson
928728    column=column_family:PHONE, timestamp=9223372036854775807, value=21190182420
928729    column=column_family:AMOUNT, timestamp=9223372036854775807, value=3245234
928729    column=column_family:CREDIT_CARD, timestamp=9223372036854775807, value=173483871743706
928729    column=column_family:NAME, timestamp=9223372036854775807, value=Hultgren Caylor
928729    column=column_family:PHONE, timestamp=9223372036854775807, value=27995164409
928730    column=column_family:AMOUNT, timestamp=9223372036854775807, value=85447788
928730    column=column_family:CREDIT_CARD, timestamp=9223372036854775807, value=7277097339102446
928730    column=column_family:NAME, timestamp=9223372036854775807, value=Lorie Garvey
928730    column=column_family:PHONE, timestamp=9223372036854775807, value=70201301198
928731    column=column_family:AMOUNT, timestamp=9223372036854775807, value=2300567
928731    column=column_family:CREDIT_CARD, timestamp=9223372036854775807, value=5730496842473502
928731    column=column_family:NAME, timestamp=9223372036854775807, value=Bourne Jose
928731    column=column_family:PHONE, timestamp=9223372036854775807, value=87122238232
928732    column=column_family:AMOUNT, timestamp=9223372036854775807, value=85447788
928732    column=column_family:CREDIT_CARD, timestamp=9223372036854775807, value=7277097339102446
928732    column=column_family:NAME, timestamp=9223372036854775807, value=Lorie Garvey
928732    column=column_family:PHONE, timestamp=9223372036854775807, value=70201301198
```

Based on the unprotected data that appears, Fred can analyze the following:

- Amounts spent by users
- Fraudulent credit card transactions – In the sample, the user Lorie Garvey might be a fraudulent user as the transaction amount of 85447788 is repeated thrice across the transactions.
- Number of repeat users – In the sample, there are three repeat users.

The HBase sample can be extended to perform actual analysis and isolate the amounts spent, fraudulent transactions, and repeat users.

➤ **To exit the HBase shell (as Fred):**

This command exits the HBase shell.

```
hbase#> exit
```

➤ **To logout the user Fred:**

This command logs out the user Fred.

```
#> exit
```

12.9 Protecting Data using Impala

You can utilize the Protegrity Impala UDFs to secure data for Impala. The Protegrity Impala UDFs are loaded during installation. While inserting data to Impala tables, or retrieving data from protected Impala table columns, you can call Protegrity Impala UDFs.

The Protegrity Impala UDFs can protect and unprotect the data as defined by the Data security policy.

For more information on the list of available Protegrity Impala UDFs, refer to section *7.3 Impala UDFs*.

The following sections describe two sample use cases.

1. A basic use case to demonstrate the functionality of how basic protection and unprotection works using Protegrity Impala UDFs.
2. A role-based use case to demonstrate the different data access permissions when two users belonging to different roles are viewing the same data.

For ease of illustration, the use cases describe the following two users:

- User with ability to protect the data, thereby accessing the data in protected form.
- User with only access to a few fields from the protected data in cleartext form.

12.9.1 Basic Use Case

This section describes the commands to perform the following functions:

- Prepare the Impala environment for the sample data.
- Initiate the Impala shell.
- Create a table in Impala using the sample data.
- Display the data in the Impala table.
- Create an Impala table containing the data protected using the Protegrity Impala UDF.
- Display the Impala table which contains the protected data.
- Unprotect the protected data in the Impala table.
- Display the Impala table which contains the unprotected data.

Ensure that you login as the user *root* before performing the following tasks.

To prepare the environment for the *basic_sample_data.csv* file:

1. Assign permissions to the path where data from the *basic_sample_data.csv* file needs to be copied using the following command:
   ```
   sudo -u hdfs hadoop fs -chown root:root /tmp/basic_sample/sample/
   ```
2. Copy the data from the *basic_sample_data.csv* file into HDFS using the following command:
   ```
   sudo -u hdfs hadoop fs -put /opt/protegrity/samples/data/basic_sample_data.csv /tmp/basic_sample/sample/
   ```
3. Verify the presence of the *basic_sample_data.csv* file in the HDFS path using the following command:
   ```
   sudo -u hdfs hadoop fs -ls /tmp/basic_sample/sample/
   ```
4. Assign permissions for Impala to the path where the *basic_sample_data.csv* file is located using the following command:
   ```
   sudo -u hdfs hadoop fs -chown impala:supergroup /tmp/basic_sample/sample/
   ```

To start the Impala shell:

This command starts the Impala shell.

```
#> impala-shell
```

To create *basic_sample* table using the *basic_sample_data.csv* file:

The following commands populate the table *basic_sample* with the data from the *basic_sample_data.csv* file.

```
drop table if exists basic_sample;
create table basic_sample(ID string, NAME string, PHONE string, CREDIT_CARD string, AMOUNT string)
row format delimited fields terminated by ',';
LOAD DATA INPATH '/tmp/basic_sample/sample/' INTO TABLE basic_sample;
```

To view the data stored in the table:

This command displays the data stored in the table.

```
select * from basic_sample;
```

Result: (Original data)

```
ID    ,  NAME           , PHONE        , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 9823750987   , 376235139103947      , 6959123
928725,  Bourne Jose    , 9823350487   , 6226600538383292     , 42964354
928726,  Sorce Hatti    , 9824757883   , 6226540862865375     , 7257656
928727,  Lorie Garvey   , 9913730982   , 5464987835837424     , 85447788
928728,  Belva Beeson   , 9948752198   , 5539455602750205     , 59040774
928729,  Hultgren Caylor, 9823750987   , 376235139103947      , 3245234
928730,  Bourne Jose    , 9823350487   , 6226600538383292     , 2300567
928731,  Lorie Garvey   , 9913730982   , 5464987835837424     , 85447788
928732,  Bourne Jose    , 9823350487   , 6226600538383292     , 3096233
928733,  Hultgren Caylor, 9823750987   , 376235139103947      , 5167763
928734,  Lorie Garvey   , 9913730982   , 5464987835837424     , 85447788
```

To create *basic_sample_protected* table to store the protected data:

The following commands create the table *basic_sample_protected* to store the protected data.

```
drop table if exists basic_sample_protected;
create table basic_sample_protected (ID string, NAME string, PHONE string, CREDIT_CARD string, AMOUNT string);
```

➤ **To protect the data in the table using Impala UDFs:**

The following command ingests cleartext data from the *basic_sample* table to the *basic_sample_protected* table in protected form using Impala UDFs.

```
insert into basic_sample_protected(ID, NAME, PHONE,CREDIT_CARD,AMOUNT) select
ID,pty_stringins(NAME,'TOK_NAME'),pty_stringins(PHONE,'TOK_PHONE'),pty_stringins(CR
EDIT_CARD,'TOK_CREDIT_CARD'),pty_stringins(AMOUNT,'TOK_AMOUNT') from basic_sample;
```

➤ **To view the protected data stored in the table:**

This command displays the protected data stored in the table.

```
select * from basic_sample_protected;
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

➤ **To unprotect the data in the table using Impala UDFs:**

The following command retrieves the unprotected data using a view.

```
create view IF NOT EXISTS view1 as select
ID,pty_stringsel(NAME,'TOK_NAME'),pty_stringsel(PHONE,'TOK_PHONE'),pty_stringsel(CR
EDIT_CARD,'TOK_CREDIT_CARD'),pty_stringsel(AMOUNT,'TOK_AMOUNT') from
basic_sample_protected;
```

➤ **To view the unprotected data stored in the table:**

This command displays the unprotected data stored in the table.

```
select id as id,_c1 as name,_c2 as phone,_c3 as credit_card,_c4 as amount from
view1;
```

Result: (Unprotected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 9823750987     , 376235139103947      , 6959123
928725,  Bourne Jose     , 9823350487     , 6226600538383292     , 42964354
928726,  Sorce Hatti     , 9824757883     , 6226540862865375     , 7257656
928727,  Lorie Garvey    , 9913730982     , 5464987835837424     , 85447788
928728,  Belva Beeson    , 9948752198     , 5539455602750205     , 59040774
928729,  Hultgren Caylor, 9823750987     , 376235139103947      , 3245234
928730,  Bourne Jose     , 9823350487     , 6226600538383292     , 2300567
928731,  Lorie Garvey    , 9913730982     , 5464987835837424     , 85447788
928732,  Bourne Jose     , 9823350487     , 6226600538383292     , 3096233
928733,  Hultgren Caylor, 9823750987     , 376235139103947      , 5167763
928734,  Lorie Garvey    , 9913730982     , 5464987835837424     , 85447788
```

12.9.2 Role-based Use Cases

This section describes the following two use cases:

- Ingestion User (*John*) ingesting and protecting the data related to credit card transactions.
- Sales Analyst (*Fred*) analyzing credit card transactions to detect the following:

- Amounts spent by users
- Fraudulent transactions
- Repeat users

12.9.2.1 Protect the Credit Card Transactions

John, the Ingestion User, is able to ingest credit card transaction data into HDFS and protect the sensitive data.

Ensure that you login as the user *John* before performing the following tasks.

➤ To login with the user John:

This command logs in the user John.

```
#> su - John
```

Enter the required password for the user *John*, when prompted.

➤ To prepare the environment for the *basic_sample_data.csv* file:

Ensure that the *basic_sample_data.csv* file is present before you execute the role-based queries. Perform the following steps to prepare the environment for the *basic_sample_data.csv* file.

1. Create an *ingestion_sample* directory for the user *John* using the following command:
   ```
   hadoop fs -mkdir /tmp/basic_sample/ingestion_sample/
   ```
2. Assign the ownership of the *ingestion_sample* directory to the Impala super users group using the following command:
   ```
   sudo -u hdfs hadoop fs -chown impala:supergroup
   /tmp/basic_sample/ingestion_sample/
   ```
3. Copy the data from the *basic_sample_data.csv* file into the *ingestion_sample* directory using the following command:
   ```
   hadoop fs -put /opt/protegrity/samples/data/basic_sample_data.csv
   /tmp/basic_sample/ingestion_sample/
   ```
4. Assign permissions for the sample data in the *ingestion_sample* directory for all users using the following command:
   ```
   hadoop fs -chmod -R 777
   /tmp/basic_sample/ingestion_sample/basic_sample_data.csv
   ```

➤ To start the Impala shell (as John):

This command starts the Impala shell.

```
#> impala-shell
```

➤ To create *basic_sample* table using the *basic_sample_data.csv* file (as John):

The following commands populate the table *basic_sample* with the data from the *basic_sample_data.csv* file.

```
drop table if exists basic_sample_ingestion_user;
create table basic_sample_ingestion_user(ID string, NAME string, PHONE string,
CREDIT_CARD string, AMOUNT string)
row format delimited fields terminated by ',';
LOAD DATA INPATH '/tmp/basic_sample/ingestion_sample/' INTO TABLE
basic_sample_ingestion_user;
```

➤ **To create table using the *basic_sample_data.csv* file to store the protected data (as John):**

The following commands create the table *basic_sample_protected_ingestion_user* to store the protected data.

```
drop table if exists basic_sample_protected_ingestion_user;
create table basic_sample_protected_ingestion_user (ID string, NAME string, PHONE
string, CREDIT_CARD string, AMOUNT string);
```

➤ **To protect data in the *basic_sample_protected_ingestion_user* table using Impala UDFs (as John):**

The following protects the data in the *basic_sample_protected_ingestion_user* table using Impala UDFs.

```
insert into basic_sample_protected_ingestion_user(ID, NAME,
PHONE,CREDIT_CARD,AMOUNT) select
ID,pty_stringins(NAME,'TOK_NAME'),pty_stringins(PHONE,'TOK_PHONE'),pty_stringins(CR
EDIT_CARD,'TOK_CREDIT_CARD'),pty_stringins(AMOUNT,'TOK_AMOUNT') from
basic_sample_ingestion_user;
```

➤ **To view the protected data stored in the table (as John):**

This command displays the protected data stored in the table.

```
select * from basic_sample_protected_ingestion_user;
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

➤ **To attempt to unprotect the data stored in the *basic_sample_protected_ingestion_user* table (as John):**

This command attempts to unprotect and display the unprotected data. The user *John* will not be able to view the cleartext data as the user does not have permissions to unprotect the data.

```
create view IF NOT EXISTS view1 as select ID, pty_stringsel(NAME, 'TOK_NAME') as name,
pty_stringsel(PHONE, 'TOK_PHONE') as phone, pty_stringsel(CREDIT_CARD,
'TOK_CREDIT_CARD') as credit_card, pty_stringsel(AMOUNT, 'TOK_AMOUNT') as amount FROM
basic_sample_protected_ingestion_user;
```

```
select * from view1;
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

➤ **To logout the user John:**

This command logs out the user John.

```
#> exit
```

12.9.2.2 Perform Analysis on Credit Card Transactions

Fred, the Sales Analyst, analyzes the amounts spent by users, the number of fraudulent transactions, and the number of repeat users, using the cleartext data in the Name and Amount fields.

Ensure that you login as the user *Fred* before performing the following tasks.

➤ **To login with the user Fred:**

This command logs in the user Fred.

```
#> su - Fred
```

Enter the required password for the user *Fred*, when prompted.

➤ **To start the Impala shell (as Fred):**

This command starts the Impala shell.

```
#> impala-shell
```

➤ **To unprotect and view the cleartext data (as Fred):**

This command unprotects the protected data and displays the unprotected data using a view. The user *Fred* will be able to view the cleartext data for the Name and Amount fields only.

```
create view IF NOT EXISTS view1 as select ID, pty_stringsel(NAME, 'TOK_NAME') as name, pty_stringsel(PHONE, 'TOK_PHONE') as phone, pty_stringsel(CREDIT_CARD, 'TOK_CREDIT_CARD') as credit_card, pty_stringsel(AMOUNT, 'TOK_AMOUNT') as amount FROM basic_sample_protected_ingestion_user;

select * from view1;
```

Result: (Protected data)

```
ID    ,  NAME           , PHONE          , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 27995164409    , 173483871743706      , 6959123
928725,  Bourne Jose    , 87122238232    , 5730496842473502     , 42964354
928726,  Sorce Hatti    , 31934151773    , 6472961686603834     , 7257656
928727,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
928728,  Belva Beeson   , 21190182420    , 3411370995179337     , 59040774
928729,  Hultgren Caylor, 27995164409    , 173483871743706      , 3245234
928730,  Bourne Jose    , 87122238232    , 5730496842473502     , 2300567
928731,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
928732,  Bourne Jose    , 87122238232    , 5730496842473502     , 3096233
928733,  Hultgren Caylor, 27995164409    , 173483871743706      , 5167763
928734,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
```

Based on the unprotected data that appears, Fred can analyze the following:

- Amounts spent by users
- Fraudulent credit card transactions – In the sample, the user Lorie Garvey might be a fraudulent user as the transaction amount of 85447788 is repeated thrice across the transactions.
- Number of repeat users – In the sample, there are three repeat users.

The Impala sample can be extended to perform actual analysis and isolate the amounts spent, fraudulent transactions, and repeat users.

➤ **To logout the user Fred:**

This command logs out the user Fred.

```
#> exit
```

12.10 Protecting Data using HAWQ

You can utilize the Protegrity HAWQ UDFs to secure data. The Protegrity HAWQ UDFs need to be defined after the Big Data Protector is installed. While inserting data to HAWQ tables, or retrieving data from protected HAWQ table columns, you can call Protegrity HAWQ UDFs.

The Protegrity HAWQ UDFs can protect and unprotect the data as defined by the Data security policy.

For more information on the list of available Protegrity HAWQ UDFs, refer to section *8.3 HAWQ UDFs*.

The following sections describe two sample use cases.

1. A basic use case to demonstrate the functionality of how basic protection and unprotection works using Protegrity HAWQ UDFs.
2. A role-based use case to demonstrate the different data access permissions when two users belonging to different roles are viewing the same data.

For ease of illustration, the use cases describe the following two users:

- User with ability to protect the data, thereby accessing the data in protected form.
- User with only access to a few fields from the protected data in cleartext form.

12.10.1 Basic Use Case

This section describes the commands to perform the following functions:

- Prepare the HAWQ environment for the sample data.
- Initiate the HAWQ shell, which is a postgres shell.

- Create a table in HAWQ using the sample data.
- Display the data in the HAWQ table.
- Create an HAWQ table containing the data protected using the Protegrity HAWQ Protect UDFs.
- Display the HAWQ table which contains the protected data.
- Unprotect the protected data in the HAWQ table using the Protegrity HAWQ Unprotect UDFs.
- Display the HAWQ table which contains the unprotected data.

Ensure that you login as the user *gpadmin* before performing the following tasks.

To start the PostgreSQL shell for HAWQ:

The following command starts the PostgreSQL shell.

```
#> psql -h <HAWQ_Master_Hostname> -p 5432
```

The following command connects to the gpadmin database.

```
\c gpadmin
```

To create the table *basic_sample* with the required number of columns:

The following command creates the table *basic_sample* with five columns, as required by the *basic_sample_data.csv* file.

```
create table basic_sample (ID varchar, NAME varchar, PHONE varchar, CREDIT_CARD
varchar, AMOUNT varchar) distributed randomly;
```

To create *basic_sample* table using the *basic_sample_data.csv* file:

The following command populates the table *basic_sample* with the data from the *basic_sample_data.csv* file.

```
\copy basic_sample from '/opt/protegrity/samples/data/basic_sample_data.csv' with
delimiter ','
```

To view the data stored in the table:

This command displays the data stored in the table.

```
select * from basic_sample order by ID;
```

Result: (Original data)

```
ID    ,  NAME           , PHONE         , CREDIT_CARD           , AMOUNT
928724,  Hultgren Caylor, 9823750987    , 376235139103947       , 6959123
928725,  Bourne Jose    , 9823350487    , 6226600538383292      , 42964354
928726,  Sorce Hatti    , 9824757883    , 6226540862865375      , 7257656
928727,  Lorie Garvey   , 9913730982    , 5464987835837424      , 85447788
928728,  Belva Beeson   , 9948752198    , 5539455602750205      , 59040774
928729,  Hultgren Caylor, 9823750987    , 376235139103947       , 3245234
928730,  Bourne Jose    , 9823350487    , 6226600538383292      , 2300567
928731,  Lorie Garvey   , 9913730982    , 5464987835837424      , 85447788
928732,  Bourne Jose    , 9823350487    , 6226600538383292      , 3096233
928733,  Hultgren Caylor, 9823750987    , 376235139103947       , 5167763
928734,  Lorie Garvey   , 9913730982    , 5464987835837424      , 85447788
```

To create *basic_sample_protected* table to store the protected data:

The following commands create the table *basic_sample_protected* to store the protected data.

```
drop table if exists basic_sample_protected;
create table basic_sample_protected (ID  varchar, NAME varchar, PHONE varchar,
CREDIT_CARD varchar, AMOUNT varchar) distributed randomly;
```

➤ **To protect the data in the table using HAWQ UDFs:**

The following command ingests cleartext data from the *basic_sample* table to the *basic_sample_protected* table in protected form using HAWQ UDFs.

```
insert into basic_sample_protected(ID, NAME, PHONE, CREDIT_CARD, AMOUNT) select ID,
pty_varcharins(NAME,'TOK_NAME'), pty_varcharins(PHONE,'TOK_PHONE'),
pty_varcharins(CREDIT_CARD,'TOK_CREDIT_CARD'), pty_varcharins(AMOUNT,'TOK_AMOUNT')
from basic_sample;
```

➤ **To view the protected data stored in the table:**

This command displays the protected data stored in the table.

```
select * from basic_sample_protected order by ID;
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

➤ **To unprotect the data in the table using HAWQ UDFs:**

The following command retrieves and displays the unprotected data.

```
create view v1 as select ID as ID, pty_varcharsel(NAME,'TOK_NAME') as NAME,
pty_varcharsel(PHONE,'TOK_PHONE') as PHONE,
pty_varcharsel(CREDIT_CARD,'TOK_CREDIT_CARD') as CREDIT_CARD,
pty_varcharsel(AMOUNT,'TOK_AMOUNT') as AMOUNT from basic_sample_protected;
```

➤ **To view the unprotected data stored in the table:**

This command displays the unprotected data stored in the table.

```
select * from v1 order by ID;
```

Result: (Unprotected data)

```
ID    ,  NAME           , PHONE        , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 9823750987   , 376235139103947      , 6959123
928725,  Bourne Jose    , 9823350487   , 6226600538383292     , 42964354
928726,  Sorce Hatti    , 9824757883   , 6226540862865375      , 7257656
928727,  Lorie Garvey   , 9913730982   , 5464987835837424      , 85447788
928728,  Belva Beeson   , 9948752198   , 5539455602750205      , 59040774
928729,  Hultgren Caylor, 9823750987   , 376235139103947      , 3245234
928730,  Bourne Jose    , 9823350487   , 6226600538383292     , 2300567
928731,  Lorie Garvey   , 9913730982   , 5464987835837424      , 85447788
928732,  Bourne Jose    , 9823350487   , 6226600538383292     , 3096233
928733,  Hultgren Caylor, 9823750987   , 376235139103947      , 5167763
928734,  Lorie Garvey   , 9913730982   , 5464987835837424      , 85447788
```

12.10.2 Role-based Use Cases

This section describes the following two use cases:

- Ingestion User (*John*) ingesting and protecting the data related to credit card transactions.
- Sales Analyst (*Fred*) analyzing credit card transactions to detect the following:

- Amounts spent by users
- Fraudulent transactions
- Repeat users

Ensure that you login as the user *gpadmin* before performing the following tasks.

➤ **To prepare the HAWQ environment for the users *John* and *Fred*:**

1. Update the *pg_hba.conf* file to include the users *John* and *Fred*.
2. Restart the HAWQ services.
3. Start the Psql shell.
4. Login to the database as *gpadmin* using the following command.
   ```
   \c gpadmin
   ```
5. Create the roles and databases for the user *John* by performing the following steps.
   a) Create the user *John* with the required password using the following command.
   ```
   create user “John”  password 'protegrity';
   ```
   b) Create a database named *John* using the following command.
   ```
   create database “John” with owner “John”;
   ```
6. Create the roles and databases for the user *Fred* by performing the following steps.
   a) Create the user *Fred* with the required password using the following command.
   ```
   create user “Fred”  password 'protegrity';
   ```
   b) Create a database named *Fred* using the following command.
   ```
   create database “Fred” with owner “Fred”;
   ```

12.10.2.1 Protect the Credit Card Transactions

John, the Ingestion User, is able to ingest credit card transaction data into HDFS and protect the sensitive data.

Ensure that you login as the user *John* before performing the following tasks.

➤ **To login with the user John:**

This command logs in the user John.

```
#> su - John
```

Enter the required password for the user *John*, when prompted.

➤ **To start the PostgreSQL shell for HAWQ (as John):**

The following command starts the PostgreSQL shell.

```
#> psql -h <HAWQ_Master_Hostname> -p 5432
```

The following command connects to the gpadmin database.

```
\c gpadmin
```

➤ **To create *basic_sample_ingestion_user* table using the *basic_sample_data.csv* file (as John):**

The following commands populate the table *basic_sample_ingestion_user* with data from the *basic_sample_data.csv* file.

```
drop table if exists basic_sample_ingestion_user;
```

```
create table basic_sample_ingestion_user (ID varchar, NAME varchar, PHONE varchar,
CREDIT_CARD varchar, AMOUNT varchar) distributed randomly;
\copy basic_sample_ingestion_user from
'/opt/protegrity/samples/data/basic_sample_data.csv' with delimiter ','
```

➤ **To view the data stored in the table:**

This command displays the data stored in the table.

```
select * from basic_sample_ingestion_user order by ID;
```

Result: (Original data)

```
ID    ,  NAME           , PHONE        , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 9823750987   , 376235139103947      , 6959123
928725,  Bourne Jose    , 9823350487   , 6226600538383292     , 42964354
928726,  Sorce Hatti    , 9824757883   , 6226540862865375      , 7257656
928727,  Lorie Garvey   , 9913730982   , 5464987835837424      , 85447788
928728,  Belva Beeson   , 9948752198   , 5539455602750205      , 59040774
928729,  Hultgren Caylor, 9823750987   , 376235139103947      , 3245234
928730,  Bourne Jose    , 9823350487   , 6226600538383292     , 2300567
928731,  Lorie Garvey   , 9913730982   , 5464987835837424      , 85447788
928732,  Bourne Jose    , 9823350487   , 6226600538383292     , 3096233
928733,  Hultgren Caylor, 9823750987   , 376235139103947      , 5167763
928734,  Lorie Garvey   , 9913730982   , 5464987835837424      , 85447788
```

➤ **To create *basic_sample_protected_ingestion_user* table to store the protected data (as John):**

The following commands create the table *basic_sample_protected_ingestion_user* to store the protected data.

```
drop table if exists basic_sample_protected_ingestion_user;
create table basic_sample_protected_ingestion_user (ID  varchar, NAME varchar,
PHONE varchar, CREDIT_CARD varchar, AMOUNT varchar) distributed randomly;
```

➤ **To grant permissions to the *basic_sample_protected_ingestion_user* table for the user Fred (as John):**

The following command grants the permissions for the table *basic_sample_protected_ingestion_user* to the user *Fred*.

```
grant all on basic_sample_protected_ingestion_user to "Fred";
```

➤ **To protect data in the *basic_sample_protected_ingestion_user* table using HAWQ UDFs (as John):**

The following protects the data in the *basic_sample_protected_ingestion_user* table using HAWQ UDFs.

```
insert into basic_sample_protected_ingestion_user(ID, NAME, PHONE, CREDIT_CARD,
AMOUNT) select ID, pty_varcharins(NAME,'TOK_NAME'),
pty_varcharins(PHONE,'TOK_PHONE'), pty_varcharins(CREDIT_CARD,'TOK_CREDIT_CARD'),
pty_varcharins(AMOUNT,'TOK_AMOUNT') from basic_sample_ingestion_user;
```

➤ **To view the protected data stored in the table (as John):**

This command displays the protected data stored in the table.

```
select * from basic_sample_protected_ingestion_user order by ID;
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

➤ **To attempt to unprotect the data stored in the *basic_sample_protected_ingestion_user* table (as John):**

This command attempts to unprotect and display the unprotected data. The user *John* will not be able to view the cleartext data as the user does not have permissions to unprotect the data.

```
select ID, pty_varcharsel(NAME,'TOK_NAME'), pty_varcharsel(PHONE,'TOK_PHONE'),
pty_varcharsel(CREDIT_CARD,'TOK_CREDIT_CARD'), pty_varcharsel(AMOUNT,'TOK_AMOUNT')
from basic_sample_protected_ingestion_user order by ID;
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

➤ **To logout the user John:**

This command logs out the user John.

```
#> exit
```

12.10.2.2 Perform Analysis on Credit Card Transactions

Fred, the Sales Analyst, analyzes the amounts spent by users, the number of fraudulent transactions, and the number of repeat users, using the cleartext data in the Name and Amount fields.

Ensure that you login as the user *Fred* before performing the following tasks.

➤ **To login with the user Fred:**

This command logs in the user Fred.

```
#> su - Fred
```

Enter the required password for the user *Fred*, when prompted.

➤ **To start the PostgreSQL shell for HAWQ (as Fred):**

The following command starts the PostgreSQL shell.

```
#> psql -h <HAWQ_Master_Hostname> -p 5432
```

The following command connects to the gpadmin database.

```
\c gpadmin
```

➤ **To unprotect and view the cleartext data (as Fred):**

These commands unprotects the protected data and displays the unprotected data using a view. The user *Fred* will be able to view the cleartext data for the Name and Amount fields only.

```
create view basic_sample_unprotected as select ID as ID,
pty_varcharsel(NAME,'TOK_NAME') as NAME, pty_varcharsel(PHONE,'TOK_PHONE') as
PHONE, pty_varcharsel(CREDIT_CARD,'TOK_CREDIT_CARD') as CREDIT_CARD,
pty_varcharsel(AMOUNT,'TOK_AMOUNT') as AMOUNT from
basic_sample_protected_ingestion_user order by ID;
select * from basic_sample_unprotected order by ID;
```

Result: (Protected data)

```
ID    ,  NAME           , PHONE          , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 27995164409    , 173483871743706      , 6959123
928725,  Bourne Jose    , 87122238232    , 5730496842473502     , 42964354
928726,  Sorce Hatti    , 31934151773    , 6472961686603834     , 7257656
928727,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
928728,  Belva Beeson   , 21190182420    , 3411370995179337     , 59040774
928729,  Hultgren Caylor, 27995164409    , 173483871743706      , 3245234
928730,  Bourne Jose    , 87122238232    , 5730496842473502     , 2300567
928731,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
928732,  Bourne Jose    , 87122238232    , 5730496842473502     , 3096233
928733,  Hultgren Caylor, 27995164409    , 173483871743706      , 5167763
928734,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
```

Based on the unprotected data that appears, Fred can analyze the following:

- Amounts spent by users
- Fraudulent credit card transactions – In the sample, the user Lorie Garvey might be a fraudulent user as the transaction amount of 85447788 is repeated thrice across the transactions.
- Number of repeat users – In the sample, there are three repeat users.

The HAWQ sample can be extended to perform actual analysis and isolate the amounts spent, fraudulent transactions, and repeat users.

➤ **To logout the user Fred:**

This command logs out the user Fred.

```
#> exit
```

12.11 Protecting Data using Spark

A Spark job in a Hadoop cluster involves sensitive data. You can use the Protegrity Spark protector APIs to protect data when it is saved or retrieved from a protected source.

The Protegrity Spark protector APIs can protect and unprotect the data as defined by the Data security policy.

For more information on the list of available Protegrity Spark protector APIs, refer to section *9.3 Spark APIs*.

The following sections describe two sample use cases.

1. A basic use case to demonstrate the functionality of how basic protection and unprotection works using Protegrity Spark protector APIs.
2. A role-based use case to demonstrate the different data access permissions when two users belonging to different roles are viewing the same data.

For ease of illustration, the use cases describe the following two users:

- User with ability to protect the data, thereby accessing the data in protected form.
- User with only access to a few fields from the protected data in cleartext form.

12.11.1 Basic Use Case

This section describes the commands to perform the following functions:

- Display the original data as is.
- Protect the original data.
- Display the data protected using the Protegrity Spark protector API.
- Unprotect the protected data.
- Display the unprotected data.

Ensure that you login as the user *root* before performing the following tasks.

➤ **To view the original data:**

This command displays the sample data as is.

```
hadoop fs -cat /tmp/basic_sample/sample/basic_sample_data.csv
```

Result: (Original data)

```
ID    ,  NAME           , PHONE         , CREDIT_CARD           , AMOUNT
928724,  Hultgren Caylor, 9823750987    , 376235139103947       , 6959123
928725,  Bourne Jose    , 9823350487    , 6226600538383292      , 42964354
928726,  Sorce Hatti    , 9824757883    , 6226540862865375      , 7257656
928727,  Lorie Garvey   , 9913730982    , 5464987835837424      , 85447788
928728,  Belva Beeson   , 9948752198    , 5539455602750205      , 59040774
928729,  Hultgren Caylor, 9823750987    , 376235139103947       , 3245234
928730,  Bourne Jose    , 9823350487    , 6226600538383292      , 2300567
928731,  Lorie Garvey   , 9913730982    , 5464987835837424      , 85447788
928732,  Bourne Jose    , 9823350487    , 6226600538383292      , 3096233
928733,  Hultgren Caylor, 9823750987    , 376235139103947       , 5167763
928734,  Lorie Garvey   , 9913730982    , 5464987835837424      , 85447788
```

➤ **To protect the data:**

This command protects the sample data. The data in the Name, Phone, Credit card, and Amount fields is protected.

```
./spark-submit --master yarn --class com.protegrity.samples.spark.ProtectData
/opt/protegrity/samples/spark/lib/spark_protector_demo.jar
hdfs://NAMENODE_HOST:8020/tmp/basic_sample/sample/basic_sample_data.csv
hdfs://NAMENODE_HOST:8020/tmp/basic_sample/sample/protected_data
```

➤ **To view the protected data:**

This command displays the protected data.

```
hadoop fs –cat /tmp/basic_sample/sample/protected_data/part*
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

➤ **To unprotect the data:**

This command unprotects the protected data.

```
./spark-submit --master yarn --class com.protegrity.samples.spark.UnProtectData
/opt/protegrity/samples/spark/lib/spark_protector_demo.jar

hdfs://NAMENODE_HOST:8020/tmp/basic_sample/sample/protected_data

hdfs://NAMENODE_HOST:8020/tmp/basic_sample/sample/unprotected_data
```

➤ **To view the unprotected data:**

This command displays the unprotected data.

```
hadoop fs -cat /tmp/basic_sample/sample/unprotected_data/part*
```

Result: (Unprotected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 9823750987     , 376235139103947      , 6959123
928725,  Bourne Jose     , 9823350487     , 6226600538383292     , 42964354
928726,  Sorce Hatti     , 9824757883     , 6226540862865375     , 7257656
928727,  Lorie Garvey    , 9913730982     , 5464987835837424     , 85447788
928728,  Belva Beeson    , 9948752198     , 5539455602750205     , 59040774
928729,  Hultgren Caylor, 9823750987     , 376235139103947      , 3245234
928730,  Bourne Jose     , 9823350487     , 6226600538383292     , 2300567
928731,  Lorie Garvey    , 9913730982     , 5464987835837424     , 85447788
928732,  Bourne Jose     , 9823350487     , 6226600538383292     , 3096233
928733,  Hultgren Caylor, 9823750987     , 376235139103947      , 5167763
928734,  Lorie Garvey    , 9913730982     , 5464987835837424     , 85447788
```

12.11.2 Role-based Use Cases

This section describes the following two use cases:

- Ingestion User (*John*) ingesting and protecting the data related to credit card transactions.
- Sales Analyst (*Fred*) analyzing credit card transactions to detect the following:
  - Amounts spent by users
  - Fraudulent transactions
  - Repeat users

12.11.2.1 Protect the Credit Card Transactions

John, the Ingestion User, is able to ingest credit card transaction data into HDFS, and protect the sensitive data.

Ensure that you login as the user *John* before performing the following tasks.

➤ **To login with the user John:**

This command logs in the user John.

```
#> su – John
```

Enter the required password for the user *John*, when prompted.

➤ **To view the original data (as John):**

This command displays the sample data as is.

```
hadoop fs -cat /tmp/basic_sample/sample/basic_sample_data.csv
```

Result: (Original data)

```
ID    ,  NAME           , PHONE        , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 9823750987   , 376235139103947      , 6959123
928725,  Bourne Jose    , 9823350487   , 6226600538383292     , 42964354
928726,  Sorce Hatti    , 9824757883   , 6226540862865375     , 7257656
928727,  Lorie Garvey   , 9913730982   , 5464987835837424     , 85447788
928728,  Belva Beeson   , 9948752198   , 5539455602750205     , 59040774
928729,  Hultgren Caylor, 9823750987   , 376235139103947      , 3245234
928730,  Bourne Jose    , 9823350487   , 6226600538383292     , 2300567
928731,  Lorie Garvey   , 9913730982   , 5464987835837424     , 85447788
928732,  Bourne Jose    , 9823350487   , 6226600538383292     , 3096233
928733,  Hultgren Caylor, 9823750987   , 376235139103947      , 5167763
928734,  Lorie Garvey   , 9913730982   , 5464987835837424     , 85447788
```

➤ **To protect the data (as John):**

The following command deletes the existing *protected_data* directory.

```
hadoop fs –rm –R /tmp/basic_sample/sample/protected_data
```

This command protects the sample data. The sensitive data is protected.

```
./spark-submit --master yarn --class com.protegrity.samples.spark.ProtectData
/opt/protegrity/samples/spark/lib/spark_protector_demo.jar
hdfs://NAMENODE_HOST:8020/tmp/basic_sample/sample/basic_sample_data.csv
hdfs://NAMENODE_HOST:8020/tmp/basic_sample/sample/ingestion_user_protected_mapred_d
ata
```

➤ **To view the protected data (as John):**

This command displays the protected data. The sensitive data appears in tokenized form.

```
hadoop fs -cat /tmp/basic_sample/ingestion_user_protected_mapred_data/part*
```

Result: (Protected data)

```
ID    ,  NAME           , PHONE        , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409  , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232  , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773  , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198  , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420  , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409  , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232  , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198  , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232  , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409  , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198  , 7277097339102446     , 945396991
```

➤ **To attempt to unprotect the data (as John):**

This command attempts to unprotect the protected data.

```
spark-submit --master yarn --class com.protegrity.samples.spark.UnProtectData
/opt/protegrity/samples/spark/lib/spark_protector_demo.jar

hdfs://NAMENODE_HOST:8020/tmp/basic_sample/sample/ingestion_user_protected_mapred_d
ata

hdfs://NAMENODE_HOST:8020/tmp/basic_sample/sample/ingestion_user_unprotected_mapred
_data
```

➤ **To attempt to view the unprotected data (as John):**

This command attempts to display the unprotected data. The user *John* will not be able to view the cleartext data as the user does not have permissions to unprotect the data.

```
hadoop fs -cat
/tmp/basic_sample/sample/ingestion_user_unprotected_mapred_data/part*
```

Result: (Protected data)

```
ID    ,  NAME            , PHONE          , CREDIT_CARD          , AMOUNT
928724,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 85924227
928725,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 83764821
928726,  Lecwe 48zhNF    , 31934151773    , 6472961686603834     , 49177868
928727,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928728,  AYEmh 2CwyvX    , 21190182420    , 3411370995179337     , 976189279
928729,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 4781777
928730,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 3285956
928731,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
928732,  4h6NlN FJi9     , 87122238232    , 5730496842473502     , 4112197
928733,  EnYEwVg3 MOQxQw , 27995164409    , 173483871743706      , 63953943
928734,  X9lLP BAA8vN    , 70201301198    , 7277097339102446     , 945396991
```

➤ **To logout the user John:**

This command logs out the user John.

```
#> exit
```

12.11.2.2 Perform Analysis on Credit Card Transactions

Fred, the Sales Analyst, analyzes the amounts spent by users, the number of fraudulent transactions, and the number of repeat users, using the cleartext data in the Name and Amount fields.

Ensure that you login as the user *Fred* before performing the following tasks.

➤ **To login with the user Fred:**

This command logs in the user Fred.

```
#> su – Fred
```

Enter the required password for the user *Fred*, when prompted.

➤ **To unprotect the data (as Fred):**

This command unprotects the protected data.

```
./spark-submit --master yarn --class com.protegrity.samples.spark.UnProtectData
/opt/protegrity/samples/spark/lib/spark_protector_demo.jar

hdfs://NAMENODE_HOST:8020/tmp/basic_sample/sample/ingestion_user_protected_mapred_d
ata

hdfs://NAMENODE_HOST:8020/tmp/basic_sample/sample/ingestion_user_unprotected_mapred
_data
```

➤ **To view the unprotected data (as Fred):**

This command displays the unprotected data. The user *Fred* will be able to view the cleartext data for the Name and Amount fields only.

```
hadoop fs -cat
/tmp/basic_sample/sample/ingestion_user_unprotected_mapred_data/part*
```

Result: (Protected data)

```
ID    ,  NAME           , PHONE          , CREDIT_CARD          , AMOUNT
928724,  Hultgren Caylor, 27995164409    , 173483871743706      , 6959123
928725,  Bourne Jose    , 87122238232    , 5730496842473502     , 42964354
928726,  Sorce Hatti    , 31934151773    , 6472961686603834     , 7257656
928727,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
928728,  Belva Beeson   , 21190182420    , 3411370995179337     , 59040774
928729,  Hultgren Caylor, 27995164409    , 173483871743706      , 3245234
928730,  Bourne Jose    , 87122238232    , 5730496842473502     , 2300567
928731,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
928732,  Bourne Jose    , 87122238232    , 5730496842473502     , 3096233
928733,  Hultgren Caylor, 27995164409    , 173483871743706      , 5167763
928734,  Lorie Garvey   , 70201301198    , 7277097339102446     , 85447788
```

Based on the unprotected data that appears, Fred can analyze the following:

- Amounts spent by users.
- Fraudulent credit card transactions – In the sample, the user Lorie Garvey might be a fraudulent user as the transaction amount of 85447788 is repeated thrice across the transactions.
- Number of repeat users – In the sample, there are three repeat users.

The Spark protector sample can be extended to perform actual analysis and isolate the amounts spent, fraudulent transactions, and repeat users.

➤ **To logout the user Fred:**

This command logs out the user Fred.

```
#> exit
```

12.11.3 Sample Code Usage for Spark (Java)

The Spark protector sample program, described in this section, is an example on how to use the Protegrity Spark protector APIs.

The sample program utilizes the following three Java classes for protecting and unprotecting data:

- **ProtectData.java** – This main class creates the Spark context object and calls the DataLoader class for reading cleartext data.
- **UnProtectData.java** - This main class creates the Spark Context object and calls the DataLoader class for reading protected data.

- **DataLoader.java** - This loader class fetches the input from the input path, calls the *ProtectFunction* to protect the data, and stores the protected data as output in the output path. In addition, it fetches the input from the protected path, calls the *UnProtectFunction* to unprotect the data, and stores the cleartext content as output.

The following functions perform protection for every new line in the input or unprotection for every new line in the output.

- **ProtectFunction** - This class calls the Spark protector for every new line specified in the input to protect data.
- **UnProtectFunction** - This class calls the Spark protector for every new line specified in the input to unprotect data.

12.11.3.1 Main Job Class for Protect Operation – ProtectData.java

```
package com.protegrity.samples.spark;

import java.io.IOException;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;

public class ProtectData {
  public static void main(String[] args) throws IOException {

    // create a SparkContext object, which tells Spark how to access a cluster.

    JavaSparkContext sparkContext =
        new JavaSparkContext(new SparkConf());

    // create the new object for class DataLoader
    DataLoader protector = new DataLoader(sparkContext);

    // Call writeProtectedData method which read clear data from input Path i.e (args[0]) and
write

    // protected data in output path i.e(args[1])
    protector.writeProtectedData(args[0], args[1], ",");
  }
}
```

12.11.3.2 Main Job Class for Unprotect Operation – UnProtectData.java

```
package com.protegrity.samples.spark;

import java.io.IOException;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;

public class UnProtectData {
  public static void main(String[] args) throws IOException {

    // create a SparkContext object, which tells Spark how to access a cluster.
    JavaSparkContext sparkContext =
        new JavaSparkContext(new SparkConf());

    // create the new object for class DataLoader
    DataLoader protector = new DataLoader(sparkContext);

    // Call unprotectData method which read protected data from input Path i.e (args[0]) and
write

    // clear data in output path i.e(args[1])
```

```
    protector.unprotectData(args[0], args[1], ",");
  }
}
```

12.11.3.3 Utility to call Protect or Unprotect Function – DataLoader.java

```
package com.protegrity.samples.spark;

import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.hdfs.DistributedFileSystem;
import org.apache.log4j.Logger;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

/**
 * A Data loader utility for reading & writing protected and un-protected data
 */
public class DataLoader {

  private JavaSparkContext sparkContext = null;

  // Declare the Array of Data Elements which will be required to do the
protection/unprotection
  private String[] data_element_names = {"TOK_NAME", "TOK_PHONE", "TOK_CREDIT_CARD",
"TOK_AMOUNT"};

  private String appid = null;
  private static final Logger logger = Logger.getLogger(DataLoader.class);

  public DataLoader(JavaSparkContext sparkContext) {
    this.sparkContext = sparkContext;
    appid = sparkContext.getConf().getAppId();
  }

  /**
   * Writes protected data to the output path delimited by the input delimiter
   *
   * @param inputPath - path of the input employee info file
   * @param outputPath - path where the output should be saved
   * @param delim - denotes the delimiter between the fields in the file
   */
  public void writeProtectedData(String inputPath, String outputPath, String delim)
      throws IOException {

    // cleans up the output path if already present
    cleanup(outputPath);

    // read lines from the input path & create RDD
    JavaRDD<String> rdd = sparkContext.textFile(inputPath);

    // Apply Protect function on rdd and get protectedRDD
    JavaRDD<String> protectedRdd = rdd.map(new ProtectFunction(appid, delim,
       data_element_names));

    // Save protectedRDD into output path
    protectedRdd.saveAsTextFile(outputPath);
  }

  /**
   * Reads protected data from the input path delimited by the input delimiter
   *
   * @param inputPath - path of the protected employee data
```

```
     * @param outputPath - output path where unprotected data should be stored.
     * @param delim
     */
    public void unprotectData(String inputPath, String outputPath, String delim) throws
        IOException {
      cleanup(outputPath);

      // read lines from the input path & create RDD
      JavaRDD<String> rdd = sparkContext.textFile(inputPath);

      // Apply Unprotect function on rdd and get original data

      JavaRDD<String> unprotectedRdd =
          rdd.map(new UnProtectFunction(appid, delim, data_element_names));

      // Save unprotectedRDD into output path
      unprotectedRdd.saveAsTextFile(outputPath);
    }

    /**
     * Deletes the output if present before spark writes the output to the given path
     *
     * @param output
     */
    private void cleanup(String output) throws IOException {
      DistributedFileSystem dfs = new DistributedFileSystem();
      try {
        dfs.initialize(new URI(output), sparkContext.hadoopConfiguration());
      } catch (URISyntaxException e) {
        logger.warn(e);
      }
      if (dfs.delete(new Path(output), true)) {
        dfs.close();
        logger.info("Cleanup: Deleted HDFS output file - " + output);
      } else
        logger.warn("Failed to delete output file!");
    }
  }
```

12.11.3.4 ProtectFunction.java

```
package com.protegrity.samples.spark;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.spark.api.java.function.Function;

import com.protegrity.spark.Protector;
import com.protegrity.spark.PtySparkProtector;
import com.protegrity.spark.PtySparkProtectorException;

public class ProtectFunction implements Function<String, String> {

  private String delim = null;
  private Protector protector = null;
  private String[] dataElement = null;
  private static final long serialVersionUID = 5693013187892705446L;

   // Initialize the spark Protector i.e ptySparkProtector in default constructor
  public ProtectFunction(String appid, String delim, String[] dataElement) throws IOException
{
    this.delim = delim;
    this.dataElement = dataElement;`
```

```
    // create the new object for class ptySparkProtector
    protector = new PtySparkProtector(appid);
  }

  @Override
  public String call(String line) throws PtySparkProtectorException {

    // splits the input seperated by delimiter in the line
    String[] splits = line.split(delim);

    // store first split in protectedString as we are not going to protect first split.
    String protectedString = splits[0];

    // Initialize input size
    String[] input = new String[splits.length];

    // Initialize output size
    String[] output = new String[splits.length];

    // Initialize errorList
    List<Integer> errorList = new ArrayList<Integer>();

    // Iterate through the splits and call protect operation
    for (int i = 1; i < splits.length; i++) {
      input[i] = splits[i];

      // To protect data,call protect method with parameter dataElement, errorList, input
array
      // and output array.output will be stored in output[]
      protector.protect(dataElement[i - 1], errorList, input, output);

      // protector.protect(dataElement[i - 1], errorList, splits, output);
      protectedString += delim + output[i];
    }

    return protectedString;
  }
}
```

12.11.3.5 UnprotectFunction.java

```
package com.protegrity.samples.spark;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.spark.api.java.function.Function;

import com.protegrity.spark.Protector;
import com.protegrity.spark.PtySparkProtector;
import com.protegrity.spark.PtySparkProtectorException;

public class UnProtectFunction implements Function<String, String> {

  private String delim = null;
  private Protector protector = null;
  private String[] dataElement = null;
  private static final long serialVersionUID = 5693013187892705446L;

  // Initialize the spark Protector i.e ptySparkProtector in default constructor
  public UnProtectFunction(String appid, String delim, String[] dataElement) throws
IOException {
```

```
        this.delim = delim;
        this.dataElement = dataElement;

        // create the new object for class ptySparkProtector
        protector = new PtySparkProtector(appid);
    }

    @Override
    public String call(String line) throws PtySparkProtectorException {

        // splits the input seperated by delimiter in the line
        String[] splits = line.split(delim);

        // store first split in unprotectedString
        String unprotectedString = splits[0];

        // Initialize input size
        String[] input = new String[splits.length];

        // Initialize output size
        String[] output = new String[splits.length];

        // Initialize errorList
        List<Integer> errorList = new ArrayList<Integer>();

        // Iterate through the splits and call unprotect operation
        for (int i = 1; i < splits.length; i++) {
          input[i] = splits[i];

          // To unprotect data,call unprotect method with parameter dataElement, errorList, input
           array
          // and output array.output will be stored in output[]
          protector.unprotect(dataElement[i - 1], errorList, input, output);
          unprotectedString += delim + output[i];
        }

        return unprotectedString;
    }

}
```

12.11.4 Sample Code Usage for Spark (Scala)

The Spark protector sample program, described in this section, is an example on how to use the Protegrity Spark protector APIs with Scala.

The sample program utilizes the following three Scala classes for protecting and unprotecting data:

- **ProtectData.scala** – This main class creates the Spark context object and calls the DataLoader class for reading cleartext data.
- **UnProtectData.scala** - This main class creates the Spark Context object and calls the DataLoader class for reading protected data.
- **DataLoader.scala -** This loader class fetches the input from the input path, calls the *ProtectFunction* to protect the data, and stores the protected data as output in the output path. In addition, it fetches the input from the protected path, calls the *UnProtectFunction* to unprotect the data, and stores the cleartext content as output.

The following functions perform protection for every new line in the input or unprotection for every new line in the output.

- **ProtectFunction** - This class calls the Spark protector for every new line specified in the input to protect data.

- **UnProtectFunction** - This class calls the Spark protector for every new line specified in the input to unprotect data.

12.11.4.1 Main Job Class for Protect Operation – ProtectData.scala

```
package com.protegrity.samples.spark.scala

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext

object ProtectData {
  def main(args: Array[String]) {
    // create a SparkContext object, which tells Spark how to access a cluster.
    val sparkContext = new SparkContext(new SparkConf())
    // create the new object for class DataLoader
    val protector = new DataLoader(sparkContext)
    // Call writeProtectedData method which read clear data from input Path i.e (args[0]) and
write data in output path after protect operation
    protector.writeProtectedData(args(0), args(1), ",")
  }
}
```

12.11.4.2 Main Job Class for Unprotect Operation – UnProtectData.scala

```
package com.protegrity.samples.spark.scala

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext

object UnProtectData {
  def main(args: Array[String]) {
    val sparkContext = new SparkContext(new SparkConf())
    val protector = new DataLoader(sparkContext)
    protector.unprotectData(args(0), args(1), ",")
  }
}
```

12.11.4.3 Utility to call Protect or Unprotect Function – DataLoader.scala

```
package com.protegrity.samples.spark.scala

import org.apache.log4j.Logger
import org.apache.spark.SparkContext

object DataLoader {
  private val logger = Logger.getLogger(classOf[DataLoader])
}
/**
 * A Data loader utility for reading & writing protected and un-protected data
 */
class DataLoader(private var sparkContext: SparkContext) {

  private var data_element_names: Array[String] = Array("TOK_NAME", "TOK_PHONE",
"TOK_CREDIT_CARD", "TOK_AMOUNT")

  private var appid: String = sparkContext.getConf.getAppId
  /**
   * Writes protected data to the output path delimited by the input delimiter
   *
   * @param inputPath - path of the input employee info file
   * @param outputPath - path where the output should be saved
   * @param delim - denotes the delimiter between the fields in the file
   */

  def writeProtectedData(inputPath: String, outputPath: String, delim: String) {
```

```
    // read lines from the input path & create RDD
    val rdd = sparkContext.textFile(inputPath)
    //import ProtectFunction
    import com.protegrity.samples.spark.scala.ProtectFunction._
    //call ProtectFunction on rdd
    rdd.ProtectFunction(delim, appid, data_element_names, outputPath)
  }

  /**
   * Reads protected data from the input path delimited by the input delimiter
   *
   * @param protectedInputPath - path of the protected employee data
   * @param unprotectedOutputPath - output path where unprotected data should be stored.
   * @param delim
   */

  def unprotectData(protectedInputPath: String, unprotectedOutputPath: String, delim: String)
{
    // read lines from the protectedInputPath & create RDD
    val protectedRdd = sparkContext.textFile(protectedInputPath)
    //import UnProtectFunction
    import com.protegrity.samples.spark.scala.UnProtectFunction._
    //call UnprotectFunction on rdd
    protectedRdd.UnprotectFunction(delim, appid, data_element_names, unprotectedOutputPath)
  }

}
```

12.11.4.4 ProtectFunction.scala

```
package com.protegrity.samples.spark.scala

import java.util.ArrayList
import org.apache.spark.rdd.RDD
import com.protegrity.spark.Protector
import com.protegrity.spark.PtySparkProtector


object ProtectFunction {
  /*Defining this class as implicit,so that we can add new functionality to an RDD on the
fly.
  implicits are lexically bounded i.e If we import this class, then only we can use it's
functions otherwise not*/
  implicit class Protect(rdd: RDD[String]) {
    def ProtectFunction(delim: String, appid: String, dataElement: Array[String],
protectoutputpath: String) =
      {
        val protectedRDD = rdd.map { line =>
          // splits the input seperated by delimiter in the line
          val splits = line.split(delim)
          // store first split in protectedString as we are not going to protect first split.
          var protectedString = splits(0)
          // Initialize input size
          val input = Array.ofDim[String](splits.length)
          // Initialize output size
          val output = Array.ofDim[String](splits.length)
          // Initialize errorList
          val errorList = new ArrayList[Integer]()
          // create the new object for class ptySparkProtector
          var protector: Protector = new PtySparkProtector(appid)
          // Iterate through the splits and call protect operation
          for (i <- 1 until splits.length) {
            input(i) = splits(i)
            // To protect data, call protect method with parameter dataElement, errorList,
input array and output array.output will be stored in output[]
            protector.protect(dataElement(i - 1), errorList, input, output)
```

```
            //Apppend output with protectedString
            protectedString += delim + output(i)
          }
          protectedString
        }

        // Save protectedRDD into output path
        protectedRDD.saveAsTextFile(protectoutputpath)

      }
  }
}
```

12.11.4.5 UnprotectFunction.scala

```
package com.protegrity.samples.spark.scala

import java.util.ArrayList
import org.apache.spark.rdd.RDD
import com.protegrity.spark.Protector
import com.protegrity.spark.PtySparkProtector


object UnProtectFunction {
  /*Defining this class as implicit,so that we can add new functionality to an RDD on the fly.
  implicits are lexically bounded i.e If we import this class, then only we can use it's
functions otherwise not*/
  implicit class Unprotect(protectedRDD: RDD[String]) {
    def  UnprotectFunction(delim:  String,  appid:  String,  dataElement:  Array[String],
unprotectoutputpath: String) =
      {
        val unprotectedRDD = protectedRDD.map { line =>
          // splits the input seperated by delimiter in the line
          val splits = line.split(delim)
          // store first split in unprotectedString
          var unprotectedString = splits(0)
          // Initialize input size
          val input = Array.ofDim[String](splits.length)
          // Initialize output size
          val output = Array.ofDim[String](splits.length)
          // Initialize errorList
          val errorList = new ArrayList[Integer]()
          // create the object for class ptySparkProtector
          var protector: Protector = new PtySparkProtector(appid)
          // Iterate through the splits and call unprotect operation
          for (i <- 1 until splits.length) {
            input(i) = splits(i)
            // To unprotect data, call unprotect method with parameter dataElement, errorList,
input array and output array.output will be stored in output[]
            protector.unprotect(dataElement(i - 1), errorList, input, output)
            //Apppend output with protectedString
            unprotectedString += delim + output(i)
          }
          unprotectedString
        }

        // Save unprotectedRDD into output path
        unprotectedRDD.saveAsTextFile(unprotectoutputpath)
      }
  }
}
```

13 Appendix: HDFSFP Demo

In this demo we perform the following tasks:

- Secure sensitive PCI and HR data that is stored in HDFS.
- Restrict access to data, using security policy specific to sensitive data.
- Track and monitor all access to sensitive data.

13.1 Roles in the Demo

The following table describes the predefined user roles in the demo.

Table 12-1 Predefined User Roles

Role / Process	Member	Description
Security Team	SamSecurity	This role sets up the security policy, creates data elements, access control list and determines who will have access to which directory.
IT Privileged	IgorIT	This role is responsible for protecting data. This role keeps the technology up.
Sales Team	SallySales	This role has full access to sales data.
HR Team	TomHR	This role has full access to employee data.
Executive Team	JohnCEO	This role has read access to all data.
Data Scientist	KerryAnalyst	This role performs analysis on sales and employee data.

13.2 HDFS Directories used in Demo

The following HDFS directories are used in the Demo:

- /company/employees
- /company/sales
- /company/public
- /company/analysis
- /company/analysis/sales
- /company/analysis/employees

13.3 User Permissions for HDFS Directories

The following table lists the user permissions for the HDFS directories in the demo.

Table 12-2 User Permissions for HDFS Directories

Directory Path	SamSecurity	IgorIT	SallySales	TomHR	JohnCEO	KerryAnalyst
/company/employees	No Access	Write Create	No Access	Read Write Create Delete	Read	Read
/company/sales	No Access	Write Create	Read Write Create	No Access	Read	Read

Directory Path	SamSecurity	IgorIT	SallySales	TomHR	JohnCEO	KerryAnalyst
			Delete			
/company/public	No Access	Write Create	Read	Read	Read	Read
/company/analysis	No Access	No Access	No Access	No Access	No Access	Read Write Create Delete
/company/analysis/sales	No Access	No Access	Read	No Access	Read	Read Write Create Delete
/company/analysis/employees	No Access	No Access	No Access	Read	Read	Read Write Create Delete

13.4 Prerequisites for the Demo

- Download the Big Data Protector.
- Install the ESA.
- Install the Big Data Protector.
- Create the following Linux OS users:
  - *SamSecurity*
  - *IgorIT*
  - *SallySales*
  - *TomHR*
  - *JohnCEO*
  - *KerryAnalyst*
- Create an unstructured policy using Policy management in ESA with the following user roles:
  - SecurityTeam - Add the user *SamSecurity* as the policy user.
  - ITPrivileaged - Add the user *IgorIT* as the policy user.
  - SalesTeam - Add the user *SallySales* as the policy user.
  - HRTeam - Add the user *TomHR* as the policy user.
  - ExecutiveTeam - Add the user *JohnCEO* as the policy user.
  - DataScientist - Add the user *KerryAnalyst* as the policy user.
- Create the following data elements with AES-256 encryption:
  - *EmployeeDE*
  - *SalesDE*
  - *AnalysisDE*
  - *SalesAnalysisDE*
  - *EmpAnalysisDE*
- Create a structured policy using Policy management in ESA with the following parameters:
  - Create a data element TRN_CUSTOMER_NAME with alpha numeric tokenization.
  - Create a data element CC_NUMBER with credit card tokenization.

- Assign the protect permission to the *DataScientist* user role for the data elements TRN_CUSTOMER_NAME and CC_NUMBER.

- Add permissions, roles and data elements mapping in the policy, as defined in the following table.

Table 12-3 Permissions, Roles and Data Elements

Role	Data Element	Unprotect	Protect	Reprotect	Delete	Create	Manage protection
ITPrivileaged	EmployeeDE	No	Yes	No	No	Yes	No
ITPrivileaged	SalesDE	No	Yes	No	No	Yes	No
ITPrivileaged	AnalysisDE	No	No	No	No	No	No
ITPrivileaged	SalesAnalysisDE	No	No	No	No	No	No
ITPrivileaged	EmpAnalysisDE	No	No	No	No	No	No
SalesTeam	EmployeeDE	No	No	No	No	No	No
SalesTeam	SalesDE	Yes	Yes	Yes	Yes	Yes	No
SalesTeam	AnalysisDE	No	No	No	No	No	No
SalesTeam	SalesAnalysisDE	Yes	No	No	No	No	No
SalesTeam	EmpAnalysisDE	No	No	No	No	No	No
HRTeam	EmployeeDE	Yes	Yes	Yes	Yes	Yes	No
HRTeam	SalesDE	No	No	No	No	No	No
HRTeam	AnalysisDE	No	No	No	No	No	No
HRTeam	SalesAnalysisDE	No	No	No	No	No	No
HRTeam	EmpAnalysisDE	Yes	No	No	No	No	No
ExecutiveTeam	EmployeeDE	Yes	No	No	No	No	No
ExecutiveTeam	SalesDE	Yes	No	No	No	No	No
ExecutiveTeam	AnalysisDE	Yes	No	No	No	No	No
ExecutiveTeam	SalesAnalysisDE	Yes	No	No	No	No	No
ExecutiveTeam	EmpAnalysisDE	Yes	No	No	No	No	No
DataScientist	EmployeeDE	Yes	No	No	No	No	No
DataScientist	SalesDE	Yes	No	No	No	No	No
DataScientist	AnalysisDE	Yes	Yes	Yes	Yes	Yes	No
DataScientist	SalesAnalysisDE	Yes	Yes	Yes	Yes	Yes	No
DataScientist	EmpAnalysisDE	Yes	Yes	Yes	Yes	Yes	No

- Deploy the policy on all nodes in the Hadoop cluster.
- Configure the *Protegrity Crypto codec*.

  For more information about configuring the *Protegrity Crypto codec*, refer to section *3.1.8.1 Configuring HDFSFP for MapReduce, v1 (MRv1)*.
- ACL entries and its permission data element used in demo are as listed in the following table.

Table 12-4 User Permissions for HDFS Directories

HDFS Path	Permission Data Element	Data Store
/company/employees	EmployeeDE	mystore
/company/sales	SalesDE	mystore
/company/analysis	AnalysisDE	mystore
/company/analysis/sales	SalesAnalysisDE	mystore
/company/analysis/employees	EmpAnalysisDE	mystore

13.5 Running the Demo

13.5.1 Protecting Existing Data in HDFS

Prerequisites:

- Install HDFSFP.
- Configure HDFSFP on the Hadoop cluster.
- Create and deploy polices.
- Create the employees directory and copy employee data into it using the following commands:
  hadoop fs -put <PROTEGRITY_DIR>/hdfsfp/demo/employee-data.csv /company/employees
  hadoop fs -put employee-travel-policy.doc /company/employees
- Start the Cache Monitor daemon with the *ITUser (IgorIT)* user
- Create the datastore on the ESA using the following command and start the Cache Refresh daemon:
  dfsdatastore -add mydatastore -host <external ip of name node> -port <Protegrity Cache port> -auth <Protegrity Cache password>

Postrequisites:

- All data inside the *company/employees* directory should get protected.

➤ **To protect existing employee data in HDFS:**

1. Add a cluster using the *dfsdatastore* utility.
2. Start the *dfscachemon* daemon on the ESA Web Interface.
3. Ensure that the Security Officer creates the ACL entry for the HDFS path */company/employees*.
4. Run the following *dfsadmin* command in the ESA CLI Manager.
   dfsadmin –protect /company/employees –permissionde EmployeeDE –datastore <Datastore name>
5. Execute the *dfsadmin* command using the following command:
   dfsadmin –activate –datastore <Datastore name>
6. Accept the **Confirm** option.
7. Verify if the data is protected using the following command:
   hadoop ptyfs -cat /company/employees/employee-data.csv | less

13.5.2 Ingesting Data into a Protected Directory

Prerequisites:

- Install HDFSFP.
- Configure HDFSFP on the Hadoop cluster.
- Create and deploy policies.
- The *ITUser* user creates the */company/sales* directory in HDFS.
- The Security Officer creates the ACL entry for the sales directory using the *dfsadmin* utility.

Postrequisites:

- The data ingested in the sales directory should get protected.

➤ **To ingest data into the protected sales directory:**

1. Login as the *IgorIT* user by using the following command.
   ```
   su – IgorIT
   ```
2. Navigate to the directory *<PROTEGRITY_DIR>/hdfsfp/demo*.
3. Run the following command as the *ITUser* user to ingest data into the sales directory:
   ```
   hadoop ptyfs -copyFromLocal sales-data.csv /company/sales
   ```
4. Verify if the path is protected using the following command:
   ```
   hadoop ptyfs –cat /company/sales/sales-data.csv | less
   ```

13.5.3 Ingesting Data into an Unprotected Public Directory

Prerequisites:

- Install HDFSFP.
- Configure HDFSFP on the Hadoop cluster.
- Create and deploy policies.
- The *ITUser* user creates the */company/public* directory in HDFS.

Postrequisites:

- Data ingested inside the public directory should not be protected.

➤ **To ingest data into an unprotected public directory:**

1. Login as *IgorIT* using the following command:
   ```
   su – IgorIT
   ```
2. Navigate to the directory */opt/protegrity/hdfsfp*.
3. Run the following command as the *ITUser* user to ingest data into the sales directory:
   ```
   hadoop ptyfs –copyFromLocal public-data.csv /company/public
   ```
4. Verify if the path is protected using the following command:
   ```
   hadoop ptyfs –cat /company/public/public-data.csv | less
   ```

13.5.4 Reading the Data by Authorized Users

Prerequisites:

- Install HDFSFP.
- Configure HDFSFP on the Hadoop cluster.
- Create and deploy policies.
- Execute section *12.5.2 Ingesting Data into a Protected Directory*.
- Create the local directory */opt/protegrity/data/employees*.

Postrequisites:

- The HR user should be able to read employee data in clear form.

➤ **To read employee data:**

1. Login as the *TomHR* user using the following command:
   su – TomHR.
2. Navigate to the directory */opt/protegrity/hdfsfp*.
3. Run the following command as the *TomHR* user to read employee data:
   hadoop ptyfs –copyToLocal /company/employees/employee-data.csv /opt/protegrity/data/employees
4. Verify if the data is unprotected using the following command:
   cat /opt/protegrity/data/employees/employee-data.csv | less

13.5.5 Reading the Data by Unauthorized Users

Prerequisites:

- Install HDFSFP.
- Configure HDFSFP on the Hadoop cluster.
- Create and deploy policies.
- Execute section *12.5.2 Ingesting Data into a Protected Directory*.
- Create the local directory */opt/protegrity/data/employees*.

Postrequisites:

- The HR user receives the *Permission Denied* error.

➤ **To read sales data:**

1. Login as the *TomHR* user using the following command:
   su – TomHR.
2. Navigate to the directory *<PROTEGRITY_DIR>/hdfsfp/demo*.
3. Run the following command as the *TomHR* user to read sales data:
   hadoop ptyfs –copyToLocal /company/sales/sales-data.csv /opt/protegrity/data/employees

13.5.6 Copying Data from One Directory to Another by Authorized Users

Prerequisites:

- Install HDFSFP.
- Configure HDFSFP on the Hadoop cluster.
- Create and deploy policies.
- Execute section *12.5.2 Ingesting Data into a Protected Directory*.

Postrequisites:

- The sales user is able to copy sales data in clear form.

➤ **To copy data from one directory to another:**

1. Login as the *SallySales* user using the following command:
   su – SallySales
2. Navigate to the directory *<PROTEGRITY_DIR>/hdfsfp/demo*.

3. Run the following command as the *SallySales* user to read sales data:
   hadoop ptyfs -cp /company/sales/sales-data.csv /company/analysis/sales
4. Verify if the data is unprotected using the following command:
   hadoop ptyfs -cat /company/analysis/sales/sales-data.csv | less

13.5.7 Copying Data from One Directory to Another by Unauthorized Users

Prerequisites:

- Install HDFSFP.
- Configure HDFSFP on the Hadoop cluster.
- Create and deploy policies.
- Execute section *12.5.1 Protecting Existing Data in HDFS*.

Postrequisites:

- The sales user should receive the *Write Permission Denied* exception for the directory */company/employees*.

➤ **To copy data from one directory to another:**

1. Login as the *SallySales* user using the following command:
   su – SallySales.
2. Navigate to the *<PROTEGRITY_DIR>/hdfsfp/demo*.
3. Run the following command as the *SallySales* user to copy data to the employees directory:
   hadoop ptyfs -put employee-list.csv /company/employees

13.5.8 Deleting Data by Authorized Users

Prerequisites:

- Install HDFSFP.
- Configure HDFSFP on Hadoop cluster.
- Create and deploy policies.
- The Security Officer creates ACL entry for the */company/analysis/sales* folder.
- Execute section *12.5.2 Ingesting Data into a Protected Directory*.

Postrequisites:

- The sales user should able to delete sales data.

➤ **To delete data:**

1. Login as the *SallySales* user using the following command:
   su – SallySales.
2. Navigate to the directory *<PROTEGRITY_DIR>/hdfsfp/demo*.
3. Run the following command as the *SallySales* user:
   hadoop ptyfs -rm /company/analysis/sales/sales-data.csv
4. Verify if the data is deleted using the following command:
   hadoop ptyfs -ls /company/analysis/sale

13.5.9 Deleting Data by Unauthorized Users

Prerequisites:

- Install HDFSFP.
- Configure HDFSFP on the Hadoop cluster.
- Create and deploy policies.
- The Security Officer creates ACL entry for the */company/analysis/sales* folder.
- Execute section *12.5.2 Ingesting Data into a Protected Directory*.

Postrequisites:

- The HR user should get the *Delete Permission Denied* exception.

➤ **To delete data:**

1. Login as the *SallySales* user using the following command:
   su – SallySales.
2. Navigate to the directory *<PROTEGRITY_DIR>/hdfsfp/demo*.
3. Run the following command as the *SallySales* user:
   hadoop ptyfs -rm /company/analysis/sales/sales-data.csv

13.5.10 Copying Data to a Public Directory by Authorized Users

Prerequisites:

- Install HDFSFP.
- Configure HDFSFP on the Hadoop cluster.
- Create and deploy policies.
- Execute section *12.5.1 Protecting Existing Data in HDFS*.

Postrequisites:

- The HR user should be able to copy employee data in clear form into the public directory.

➤ **To copy data to a public directory:**

1. Login as the *TomHR* user using the following command:
   su – TomHR.
2. Navigate to the directory *<PROTEGRITY_DIR>/hdfsfp/demo*.
3. Run the following command as the *TomHR* user:
   hadoop ptyfs -cp /company/employees/employee-travel-policy.doc /company/public
4. Verify if the data is copied using the following command:
   hadoop ptyfs -cat /company/public/employee-travel-policy.doc | less

13.5.11 Running MapReduce Job by Authorized Users

Prerequisites:

- Install HDFSFP.
- Configure HDFSFP on the Hadoop cluster.
- Create and deploy policies.
- Execute section *12.5.2 Ingesting Data into a Protected Directory*.

Postrequisites:

- The MapReduce job should be successfully completed. The output of the MapReduce job should be saved in protected form in the */company/analysis/sales* directory.

➤ **To run a MapReduce Job:**

1. Login as the *SallySales* user using the following command
   su – SallySales
2. Navigate to the directory *<PROTEGRITY_DIR>/hdfsfp/demo*.
3. Run the following command as the *SallySales* user:
   hadoop jar <PROTEGRITY_DIR>/hdfsfp/hdfsfp-x.x.x.jar com.protegrity.hadoop.fileprotector.PtyCryptoWriter /company/sales /company/analysis/sales/result.

In the MapReduce job for sales analysis using the Big Data Protector API, we tokenize customer name and credit card data.

13.5.12 Reading Data for Analysis by Authorized Users

Prerequisites:

- Install HDFSFP.
- Configure HDFSFP on the Hadoop cluster.
- Create and deploy policies.
- Execute section *12.5.11 Running MapReduce Job by Authorized Users*.
- Create the local directory */opt/protegrity/data/sales/analysis*.

Postrequisites:

- The data scientist should able to read analysis sales data in clear form with the PCI data tokenized.

➤ **To read data for analysis:**

1. Login as the *KellyAnalyst* user using the following command
   su – KellyAnalyst.
2. Navigate to the directory *<PROTEGRITY_DIR>/hdfsfp/demo*.
3. Run the following command as the *KellyAnalyst* user to read the analysis of the sales data:
   hadoop ptyfs -copyToLocal /company/analysis/sales/result/part-r-00000 /opt/protegrity/data/sales/analysis
4. Verify if the data is unprotected with the PCI data tokenized using the following command:
   cat /opt/protegrity/data/sales/analysis/part-r-00000 | less

14 Appendix: Using Hive with HDFSFP

This section describes how to use Hive with HDFSFP.

The Hive native *Load* command for loading data from any file in the local file system or HDFS to a protected Hive table is not supported.

This release supports *TEXTFILE*, *RCFile*, and *SEQUENCEFILE* formats only.

14.1 Data Used by the Samples

The *employee.csv* file used in the examples demonstrating usage of Hive with HDFSFP contains the following sample data.

```
928724,Hultgren Caylor
928725,Bourne Jose
928726,Sorce Hatti
928727,Lorie Garvey
928728,Belva Beeson
928729,Hultgren Caylor
928730,Bourne Jose
928731,Lorie Garvey
928732,Bourne Jose
928733,Hultgren Caylor
928734,Lorie Garvey
```

14.2 Ingesting Data to Hive Table

If you need to ingest data from a relational database to a Hive protected table in HDFS, then ensure that you load the data through Sqoop and use the *-D target.output.dir* parameter, as described in the following command.

```
sqoop import -D target.output.dir="/tmp/src" --fields-terminated-by ',' --driver
com.mysql.jdbc.Driver --connect jdbc:mysql://master.localdomain:3306/db --username
user1 --password protegrity --table emp --hive-import --hive-table=emp_sqp -m 1;
```

14.2.1 Ingesting Data from HDFSFP Protected External Hive Table to HDFSFP Protected Internal Hive Table

➤ **To ingest data from an HDFSFP protected external Hive table to an HDFSFP protected internal Hive table:**

1. Create a Protect ACL for the following path.
   ```
   /user/ptyitusr/ext1
   ```
2. Copy the cleartext file (*employee.csv*) into the protected staging directory (*/user/ptyitusr/ext1*) using the following commands:
   ```
   hadoop ptyfs -put employee.csv /user/ptyitusr/ext1
   hadoop fs -ls /user/ptyitusr/ext1
   hadoop fs -cat /user/ptyitusr/ext1/employee.csv
   ```
3. Create an external Hive table (*ext1*) with the data contained in the protected staging directory (*/user/ptyitusr/ext1*) using the followind commands.
   ```
   CREATE EXTERNAL TABLE ext1 (empid string, name string) ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION
   '/user/ptyitusr/ext1';
   select * from ext1;
   ```

4. Create a Protect ACL for the following path.
   <hive.metastore.warehouse.dir>/int1
5. Create an internal Hive table (*int1*) using the followind command.
   CREATE TABLE int1 (empid string, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE;
6. Insert the cleartext data from the external Hive table (*ext1*) to the internal Hive table (*int1*) in protected form using the following commands.
   SET hive.exec.compress.output=true;
   insert overwrite table int1 select empid, name from ext1;
   The data is stored in the internal Hive table *int1* in protected form.
7. Revert the value of the *hive.exec.compress.output* parameter using the following command.
   SET hive.exec.compress.output=false;
8. To view the data in the table *int1*, execute the following command.
   select * from int1;

14.2.2 Ingesting Protected Data from HDFSFP Protected Hive Table to another HDFSFP Protected Hive Table

➤ **To ingest protected data from an HDFSFP protected external Hive table to another HDFSFP protected Hive table:**

1. Create Protect ACLs for the following paths.
   /user/ptyitusr/ext1
   /user/ptyitusr/ext2
2. Copy the cleartext file (*employee.csv*) into the protected staging directory (*/user/ptyitusr/ext1*) using the following commands:
   hadoop ptyfs -put employee.csv /user/ptyitusr/ext1
   hadoop fs -ls /user/ptyitusr/ext1
   hadoop fs -cat /user/ptyitusr/ext1/employee.csv
3. Create an external table (*ext1*) with the data contained in the protected staging directory using the following commands:
   CREATE EXTERNAL TABLE ext1 (empid string, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/ext1';
   select * from ext1;
4. Create an external Hive table (*ext2*) using the following command:
   CREATE EXTERNAL TABLE ext2 (empid string, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/ext2';
5. Insert the cleartext data from the external Hive table *ext1* to the Hive table *ext2* in protected form using the following commands:
   SET hive.exec.compress.output=true;
   insert overwrite table ext2 select empid, name from ext1;
   The data is stored in the Hive table *ext2* in protected form.
6. Revert the value of the *hive.exec.compress.output* parameter using the following command.
   SET hive.exec.compress.output=false;
7. To view the data in the table *ext2*, execute the following command.
   select * from ext2;

14.3 Tokenization and Detokenization with HDFSFP

14.3.1 Verifying Prerequisites for Using Hadoop Application Protector

Ensure that the following data elements are present in data security policy and the policy is deployed on all the Big Data Protector nodes:

- ***TOK_NUM_1_3_LP*** – Length preserving Numeric tokenization data element with SLT_1_3
- ***TOK_ALPHA_2_3_LP*** – Length preserving Alpha tokenization data element with SLT_2_3

14.3.2 Ingesting Data from HDFSFP Protected External Hive Table to HDFSFP Protected Internal Hive Table in Tokenized Form

➤ **To ingest data from an HDFSFP protected external Hive table to an HDFSFP protected internal Hive table in tokenized form:**

1. Create Protect ACLs for the following paths.
   /user/ptyitusr/ext1
   <hive.metastore.warehouse.dir>/int1
2. Copy the cleartext file (*employee.csv*) into the protected staging directory (*/user/ptyitusr/ext1*) using the following commands:
   ```
   hadoop ptyfs -put employee.csv /user/ptyitusr/ext1
   hadoop fs -ls /user/ptyitusr/ext1
   hadoop fs -cat /user/ptyitusr/ext1/employee.csv
   ```
3. Create an external Hive table (*ext1*) with the data contained in the protected staging directory (*/user/ptyitusr/ext1*) using the followind commands.
   ```
   CREATE EXTERNAL TABLE ext1 (empid string, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/ext1';
   select * from ext1;
   ```
4. Create an internal Hive table (*int1*) using the followind command.
   ```
   CREATE TABLE int1 (empid string, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/ext1';
   ```
5. Insert the cleartext data from the external Hive table (*ext1*) to the internal Hive table (*int1*) in tokenized form using the following commands.
   ```
   SET hive.exec.compress.output=true;
   create temporary function ptyProtectStr AS 'com.protegrity.hive.udf.ptyProtectStr';
   insert overwrite table int1 select ptyProtectStr(empid, 'TOK_NUM_1_3_LP'), ptyProtectStr(name, 'TOK_ALPHA_2_3_LP') from ext1;
   ```
   The data is stored in the internal Hive table *int1* in tokenized form.
6. Revert the value of the *hive.exec.compress.output* parameter using the following command.
   ```
   SET hive.exec.compress.output=false;
   ```
7. To view the data in the table *int1*, execute the following command.
   ```
   select * from int1;
   ```

14.3.3 Ingesting Detokenized Data from HDFSFP Protected Internal Hive Table to HDFSFP Protected External Hive Table

➤ **To ingest detokenized data from an HDFSFP protected internal Hive table with tokenized data to an HDFSFP protected external Hive table:**

1. Create Protect ACLs for the following paths.
   /user/ptyitusr/ext1
   <hive.metastore.warehouse.dir>/int1
2. Create an external Hive table (*ext1*) with the data contained in the protected staging directory (*/user/ptyitusr/ext1*) using the followind commands.
   ```
   CREATE EXTERNAL TABLE ext1 (empid string, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/ext1';
   ```
3. Insert the tokenized data from the internal Hive table (*int1*) to the external Hive table (*ext1*) in detokenized form using the following commands.
   ```
   SET hive.exec.compress.output=true;
   create temporary function ptyUnprotectStr AS 'com.protegrity.hive.udf.ptyUnprotectStr';
   insert overwrite table ext2 select ptyUnprotectStr(empid, 'TOK_NUM_1_3_LP'), ptyUnprotectStr(name, 'TOK_ALPHA_2_3_LP') from int1;
   ```
   The tokenized data is stored in the external Hive table *ext1* in detokenized form.
4. Revert the value of the *hive.exec.compress.output* parameter using the following command.
   ```
   SET hive.exec.compress.output=false;
   ```
5. To view the data in the table *ext1*, execute the following command.
   ```
   select * from ext1;
   ```

14.3.4 Ingesting Data from HDFSFP Protected External Hive Table to Internal Hive Table not protected by HDFSFP in Tokenized Form

➤ **To ingest data from an HDFSFP protected external Hive table to an internal Hive table not protected by HDFSFP in tokenized form:**

1. Create a Protect ACL for the following path.
   /user/ptyitusr/ext1
2. Copy the cleartext file (*employee.csv*) into the protected staging directory (*/user/ptyitusr/ext1*) using the following commands:
   ```
   hadoop ptyfs -put employee.csv /user/ptyitusr/ext1
   hadoop fs -ls /user/ptyitusr/ext1
   hadoop fs -cat /user/ptyitusr/ext1/employee.csv
   ```
3. Create an external Hive table (*ext1*) with the data contained in the protected staging directory (*/user/ptyitusr/ext1*) using the followind commands.
   ```
   CREATE EXTERNAL TABLE ext1 (empid string, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/ext1';
   select * from ext1;
   ```
4. Create an internal Hive table (*int1*) using the followind command.
   ```
   CREATE TABLE int1 (empid string, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/ext1';
   ```

5. Insert the cleartext data from the external Hive table (*ext1*) to the internal Hive table (*int1*) in tokenized form using the following commands.
   ```
   SET hive.exec.compress.output=false; // Set this property to false to prevent insertion of encrypted data into the unprotected table
   create temporary function ptyProtectStr AS 'com.protegrity.hive.udf.ptyProtectStr';
   insert overwrite table int1 select ptyProtectStr(empid, 'TOK_NUM_1_3_LP'), ptyProtectStr(name, 'TOK_ALPHA_2_3_LP') from ext1;
   ```
   The data is stored in the internal Hive table *int1* in tokenized form.
6. To view the tokenized data in the table *int1*, execute the following command.
   ```
   select * from int1;
   ```

14.3.5 Ingesting Detokenized Data from Internal Hive Table not protected by HDFSFP to HDFSFP Protected External Hive Table

➤ **To ingest detokenized data from an internal Hive table with tokenized data not protected by HDFSFP to an HDFSFP protected external Hive table:**

1. Create a Protect ACL for the following path.
   ```
   /user/ptyitusr/ext1
   ```
2. Create an external Hive table (*ext1*) with the data contained in the protected staging directory (*/user/ptyitusr/ext1*) using the followind commands.
   ```
   CREATE EXTERNAL TABLE ext1 (empid string, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/ext1';
   ```
3. Insert the tokenized data from the internal Hive table (*int1*) to the external Hive table (*ext1*) in detokenized form using the following commands.
   ```
   SET hive.exec.compress.output=true;
   create temporary function ptyUnprotectStr AS 'com.protegrity.hive.udf.ptyUnprotectStr';
   insert overwrite table ext1 select ptyUnprotectStr(empid, 'TOK_NUM_1_3_LP'), ptyUnprotectStr(name, 'TOK_ALPHA_2_3_LP') from int1;
   ```
   The tokenized data is stored in the external Hive table *ext1* in detokenized form.
4. Revert the value of the *hive.exec.compress.output* parameter using the following command.
   ```
   SET hive.exec.compress.output=false;
   ```
5. To view the detokenized data in the table *ext1*, execute the following command.
   ```
   select * from ext1;
   ```

15 Appendix: Configuring Talend with HDFSFP

This section describes the procedures for configuring Talend with HDFSFP.

This section considers Talend, version 5.6 for reference.

15.1 Verifying Prerequisites before Configuring Talend with HDFSFP

Ensure that the following prerequisites are met before configuring Talend with HDFSFP:

- Install the Big Data Protector with the **HDFSFP** option in the *BDP.config* file as *Yes*.
- Deploy the unstructured policy on the cluster.

15.2 Verifying the Talend Packages

Verify if the folder *<PROTEGRITY_DIR>/etl/talend* is populated with the following folders:

- *tptyHDFSInput* – This folder contains the custom Protegrity HDFS input component.
- *tptyHDFSOutput* – This folder contains the custom Protegrity HDFS output component.
- *jars* – This folder contains the third-party JARs.
- *docs* – This folder contains the user guide for Talend.

15.3 Configuring Talend with HDFSFP

➤ **To configure Talend with HDFSFP:**

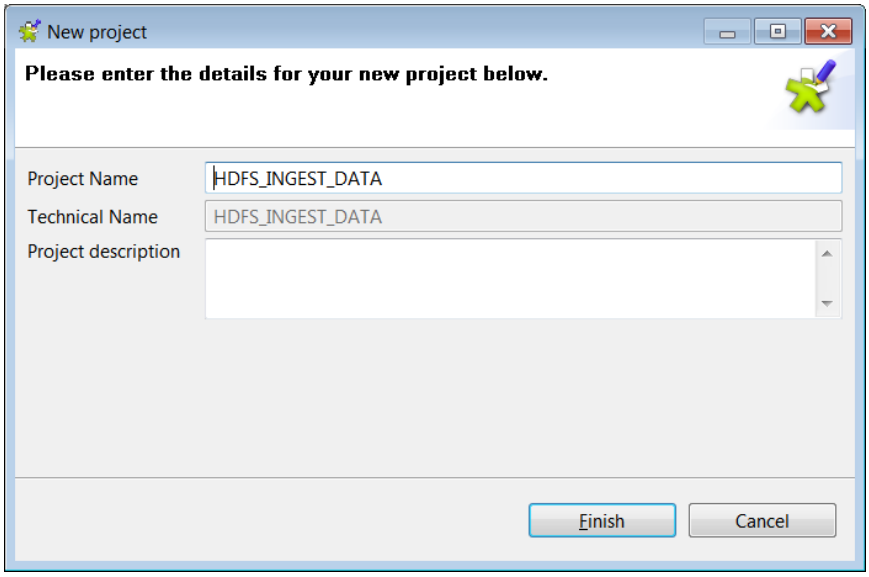
1. Login to the ESA.
2. Create the datastore.
3. Create the ACL entry for the datastore.
4. Activate the ACL entry for the datastore.
5. Create a directory for Talend using the following command.
   mkdir <PROTEGRITY_DIR>/talend/
6. Copy the *tptyHDFSInput* and *tptyHDFSOutput* folders from the *<PROTEGRITY_DIR>/etl/talend/* directory to the *<PROTEGRITY_DIR>/talend/* directory.
7. Create the *hdfsfp* directory for Talend using the following command.
   mkdir <TALEND_Installation_DIR>/hdfsfp/
8. Copy the following files files from the *<PROTEGRITY_DIR>/hdfsfp/* directory to the *<TALEND_Installation_DIR>/hdfsfp* directory.
   - *hdfsfp.jar*
   - *jedis-2.1.0.jar*
   - *beuler.properties*
   - *hdfsfp-log4j.properties*
9. Copy the following files from the *<PROTEGRITY_DIR>/defiance_xc/java/lib/* directory to the *<TALEND_Installation_DIR>/hdfsfp/* directory.
   - *xcpep2jni.jar*
   - *xcpep2jni.plm*
   - *xcpep2jni.properties*
10. Copy the following third-party JAR files from the *<PROTEGRITY_DIR>/etl/talend/jars* directory to the *<TALEND_Installation_DIR>/hdfsfp/* directory.
    - *commons-codec-1.4.jar*
    - *commons-collections-3.2.1.jar*

15.4 Starting a Project in Talend

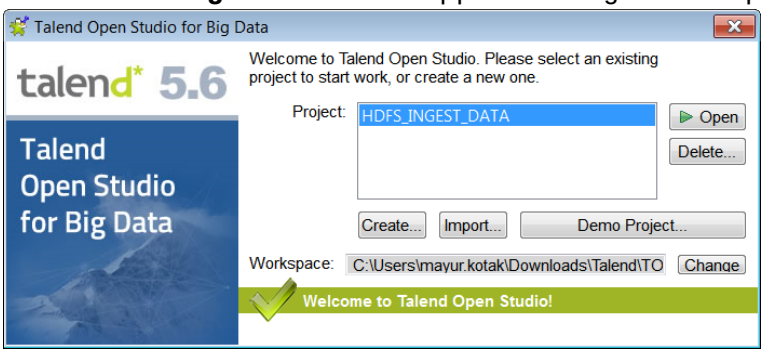
➤ **To start a project in Talend:**

1. Login to the Edgenode machine with Talend installed.
2. Execute the *TOS_BD-linux-gtk-x86.sh* script to start **Talend Open Studio for Big Data**.
   The **Talend Open Studio for Big Data** window appears.

3. In the Create a New Project box, type the following:
   `HDFS_INGEST_DATA`
4. Click **Create** to create a new project.
   The **New Project** window appears.

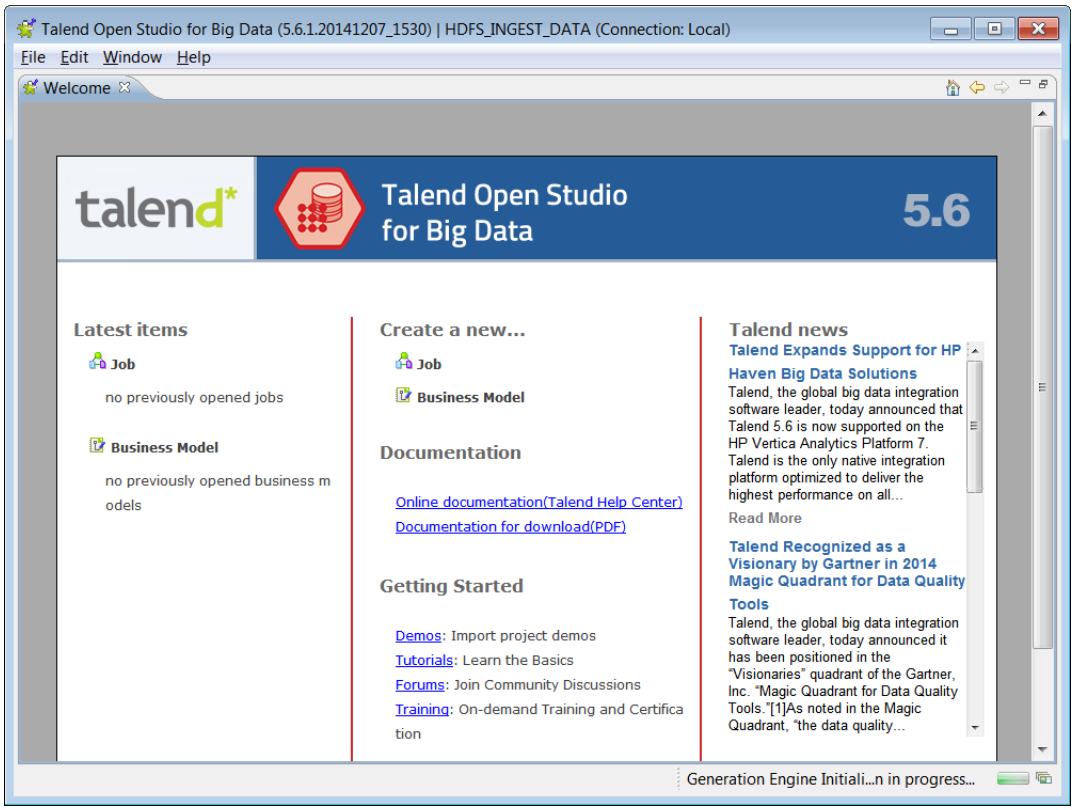


5. Click **Finish** to create the new project.
   The **Talend Open Studio for Big Data** window appears listing the new project.



6. Click **Open**.

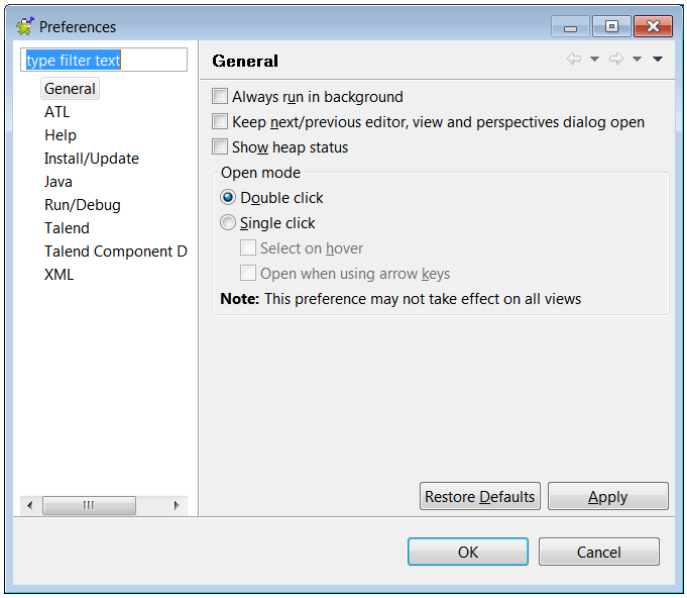
The **Talend Open Studio for Big Data** workspace appears.



15.5 Configuring the Preferences for Talend

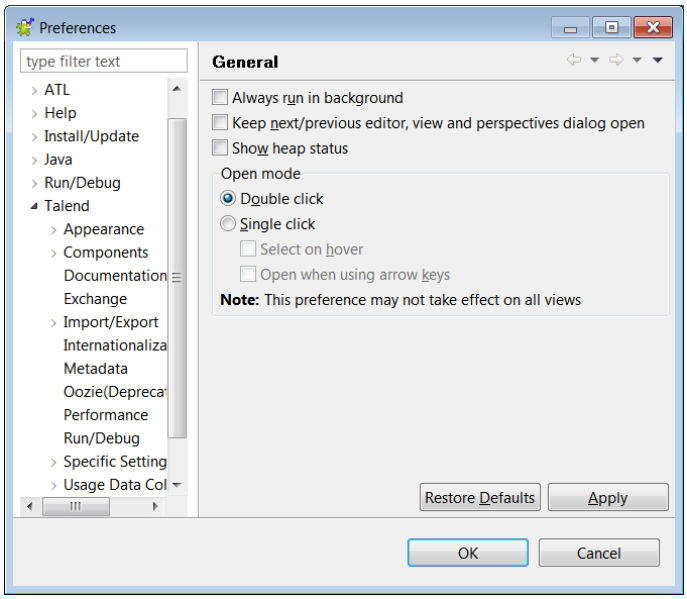
➤ **To configure the preferences for Talend:**

1. On the **Talend Open Studio for Big Data** workspace, click **Window**.
   The **Window** menu appears.
2. Click **Preferences**.
   The **Preferences** window appears.

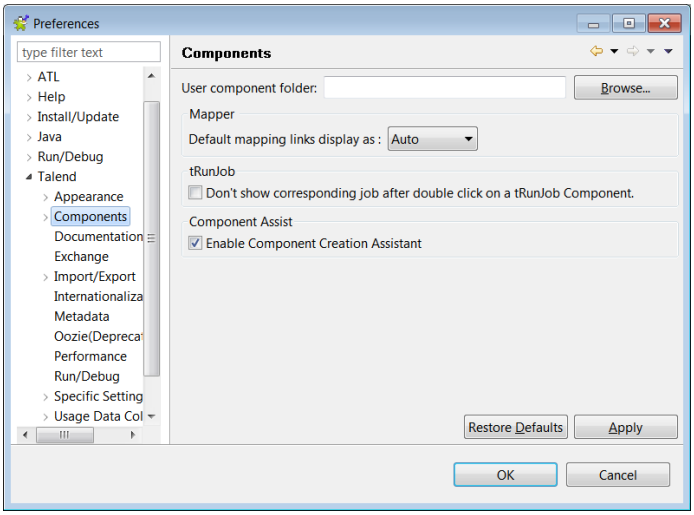


3. Click on **Talend** in the preferences pane.

The general preferences for Talend appear.



4. Click on **Components** in the preferences pane.
   The component preferences for Talend appear.



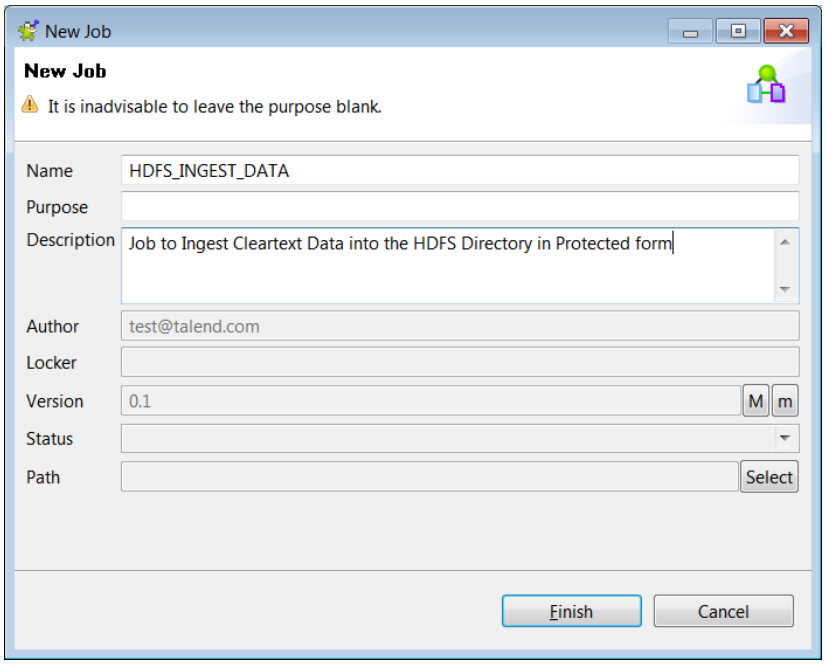
5. In the User component folder box, type the following path:
   <PROTEGRITY_DIR>/talend/
6. Click **Apply**.
7. Click **OK**.
   The preferences for Talend are updated.

15.6 Ingesting Data in the Target HDFS Directory in Protected Form

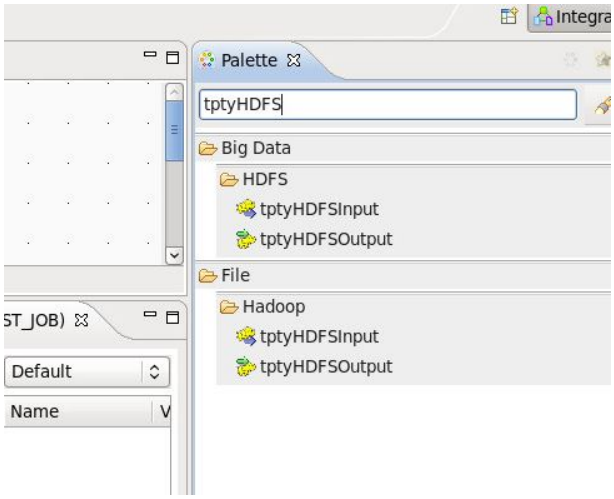
➤ **To ingest Cleartext Data into the Target HDFS Directory:**

1. Click the **Job** link under the **Create a new...** section.

The **New Job** window appears.

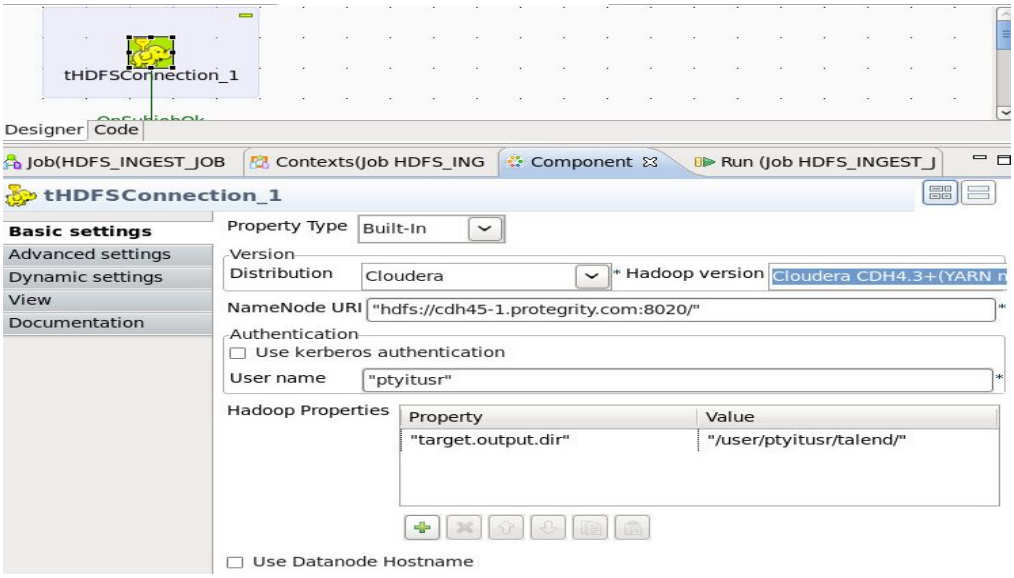


2. Type the following in the **Name** box.
   `HDFS_INGEST_DATA`
3. Enter a description in the **Description** box.
4. Click **Finish**.
   The new job in Talend is created.
5. Verify if the custom components, *tptyHDFSInput* and *tptyHDFSOutput*, are loaded into the palette by searching the name *tptyHDFS*.
   The two components, *tptyHDFSInput* and *tptyHDFSOutput* appear.

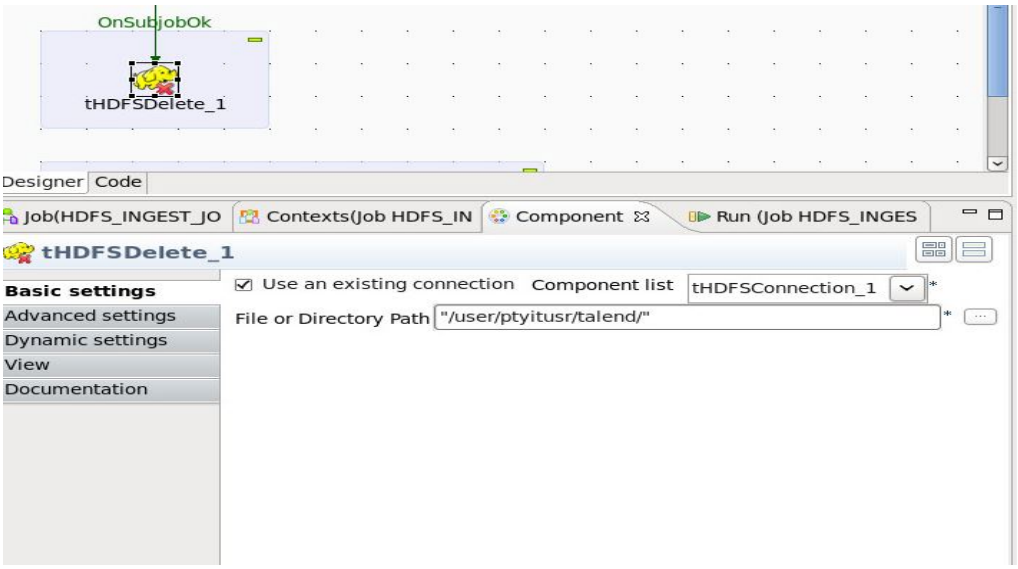


6. Double-click the *tHDFSConnection* component to create the HDFS connection.
   Enter the following properties for the connection, as required:
   - Distribution – The distribution name
   - Hadoop version – The version required for the Hadoop cluster
   - NameNode URI – The domain name and port of the Name node to connect to HDFS
   - User name – The user name to perform the HDFSFP operations. For instance, we have the user as *ptyitusr*.

- target.output.dir – The targeted HDFS directory to store the protected data. For instance, we have the HDFS directory as */user/ptyitusr/talend*.



7. Create the *tHDFSDelete* component.
   a) If the directory exists, then delete the contents of the directory.
   b) Select the *Use an existing connection* box to reuse the existing connection.



8. Create the *tFileInputDelimited* component.
   a) Create a *sample.csv* file in the local file system directory */home/ptyitusr/input/* with the following entries.

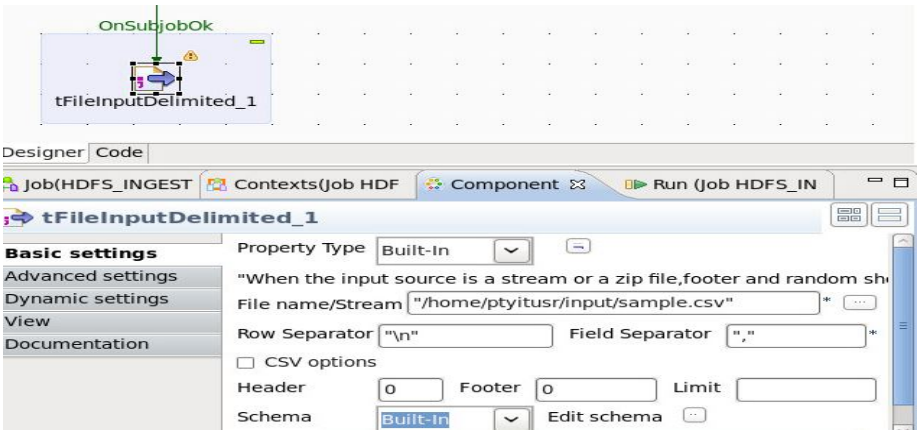
```
101,Adam,Wiley,Sales
102,Brian,Chester,Service
103,Julian,Cross,Sales
104,Dylan,Moore,Marketing
105,Chris,Murphy,Service
106,Brian,Collingwood,Service
107,Michael,Muster,Marketing
108,Miley,Rhodes,Sales
109,Chris,Coughlan,Sales
110,Aaron,King,Marketng
111,Adam,Young,Service
112,Tyler,White,Sales
```

```
113,Martin,Reeves,Service
114,Michael,Morton,Sales
```

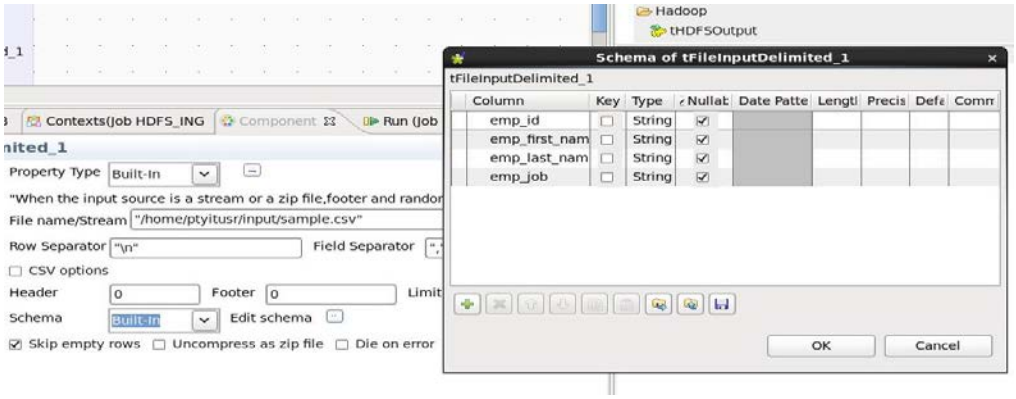
b) Enter the input directory location, as required.

c) Enter the *Row Separator* option, as required.

d) Enter the *Field Separator* option, as required.



e) Edit the schema for the input supplied, which would be required while writing data to the HDFS path.



9. Create the *tptyHDFSOutput* component.

a) Click the *Use an existing connection* box to utilize the existing connection.

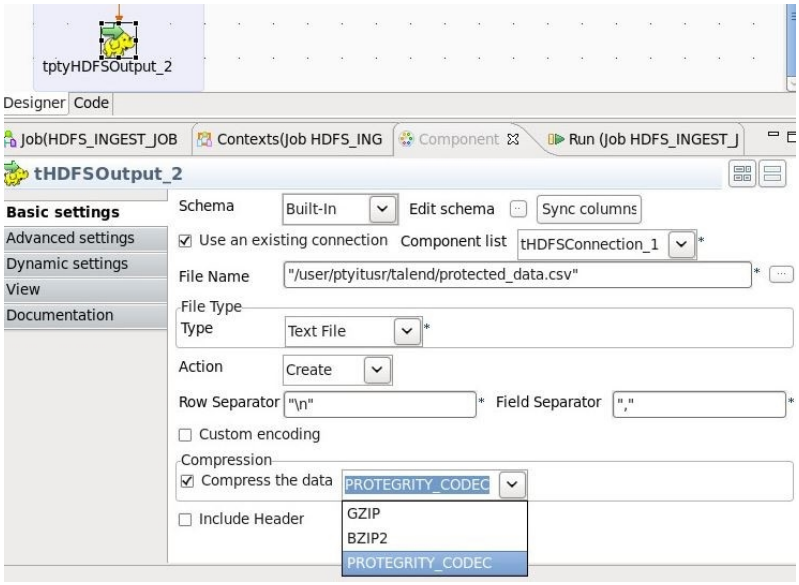
b) Enter the file location in HDFS, where the data needs to be protected and stored.

c) Enter the *Row Separator* option, as required.

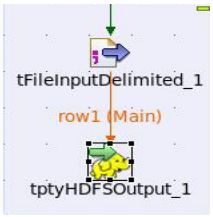
d) Enter the *Field Separator* option, as required.

e) Click *Compress the data* box.

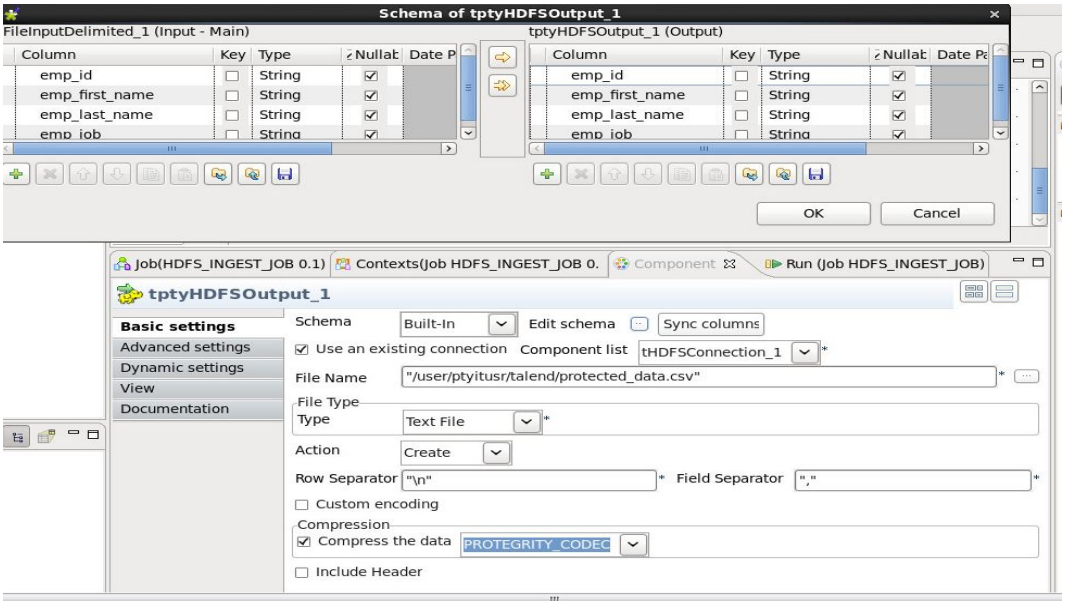
f) Select **PROTEGRITY_CODEC** from the drop down.



g) Connect the *tFileInputDelimited_1* component to the *tptyHDFSOutput_1* component as illustrated in the following figure.

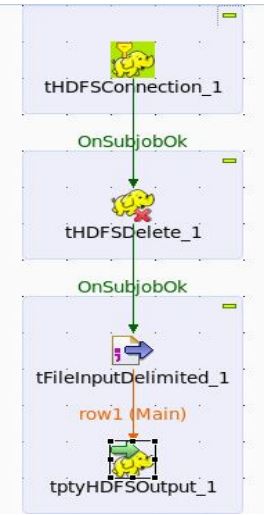


h) Click **Edit schema** to map the input supplied and output file, which will be stored in the HDFS path.

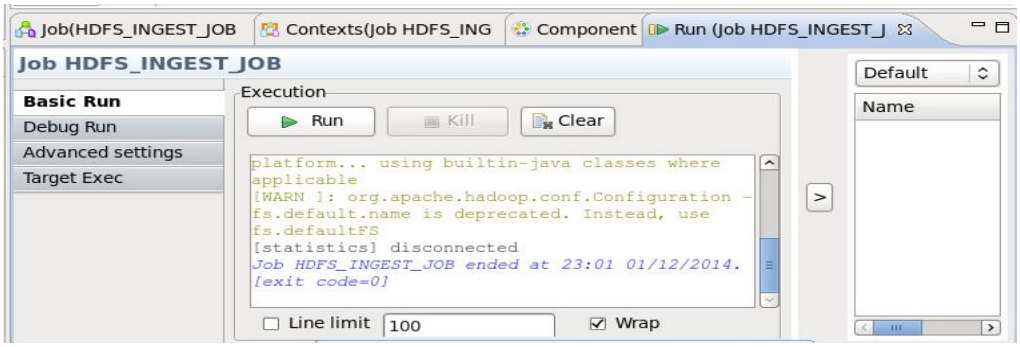


i) Click **OK**.

10. Connect all the components in the job diagram, as illustrated by the following figure.



11. Click on the *Run* tab.
    The Run tab appears.
12. Click **Run** to run the job.
    Check the job execution status in the console.



13. Login to the system with the user *ptyitusr* using the following command.
    ```
    >> su - ptyitusr
    ```
14. List the file present in the HDFS directory /user/ptyitusr/talend/ using the following command.
    ```
    >> hadoop fs -ls /user/ptyitusr/talend/
    ```
15. Read the *protected_data.csv* file from the HDFS directory */user/ptyitusr/talend/* using the following command.
    ```
    >> hadoop fs -cat /user/ptyitusr/talend/protected_data.csv
    ```
    The protected data appears.

```
[root@cdh45-1 ~]# su - ptyitusr
-sh-4.1$ hadoop fs -ls /user/ptyitusr/talend/
Found 1 items
-rw-r--r--   3 ptyitusr ptyitusrgroup        451 2014-12-01 23:56 /user/ptyitusr/talend/protected_data.csv
-sh-4.1$ hadoop fs -cat /user/ptyitusr/talend/protected_data.csv
}Ó,v@+ý)¡°0ýFÔúÍ[0dÏÃHI·T×¦·òÉØ|hÜü×"þëtÍ]×d¿Ub²Ð#±FiÍHÞæ´0K%,ñ{\LG-.àí þ
                                                                +}ípÒÓ:ÂïÅÑdbõq0dÉÉÛ=·iïäA²-6¤ ¾¼P0pÎoe{·
;úN®óý)±È6áxÌ,,/Ô
                ,Í!ÓdYán9(!¹°
                                @ÅÇîjÉ²àCí6+Ï¼{Ó"p/çýðÔò[e2ÃtãWòú«`U&©] ý¶©PÚÉ=yvÕÑs{°E(Þ(È¨;±§Ûk#Ò~¦d²â-sh-4.1$
-sh-4.1$
```

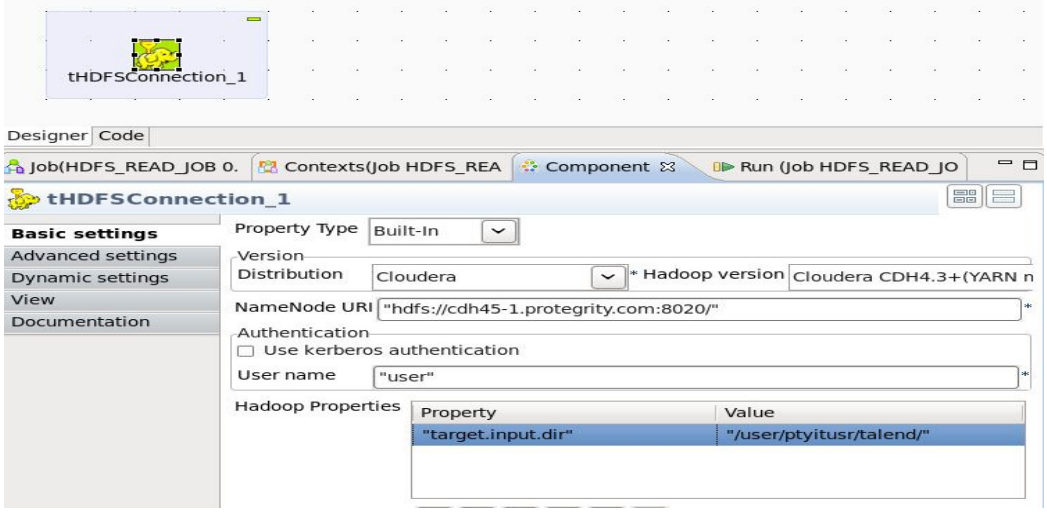
15.7 Accessing the Data from the Protected Directory in HDFS

➤ **To Access Data from the Protected Directory in HDFS:**

1. Click the **Job** link under the **Create a new...** section.

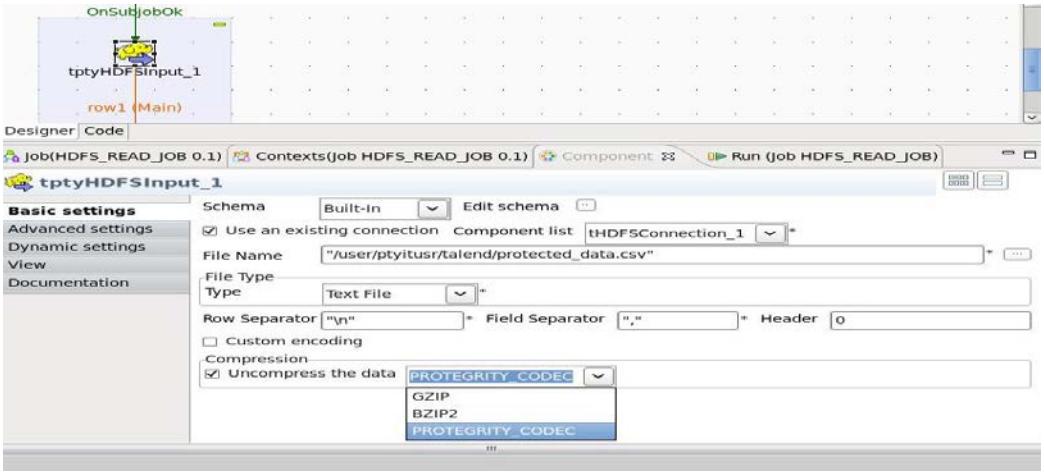
The **New Job** window appears.

2. Type the following in the **Name** box.
   HDFS_READ_DATA
3. Enter a description in the **Description** box.
4. Click **Finish**.
   The new job in Talend is created.
5. Double-click the *tHDFSConnection* component to create the HDFS connection.
   Enter the following properties for the connection, as required:
   - Distribution – The distribution name
   - Hadoop version – The version required for the Hadoop cluster
   - NameNode URI – The domain name and port of the Name node to connect to HDFS
   - User name – The user name to perform the HDFSFP operations. For instance, we have the user as *ptyitusr*.
   - target.input.dir – The targeted HDFS directory to read the protected data. For instance, we have the HDFS directory as */user/ptyitusr/talend*.



6. Create the *tptyHDFSInput* component.
   a) Select the *Use an existing connection* box to reuse the existing connection.
   b) Enter the HDFS location where the protected data is to be read, as required.
   c) Enter the *Row Separator* option, as required.
   d) Enter the *Field Separator* option, as required.
   e) Click *Uncompress the data* box.

f) Select **PROTEGRITY_CODEC** from the drop down.

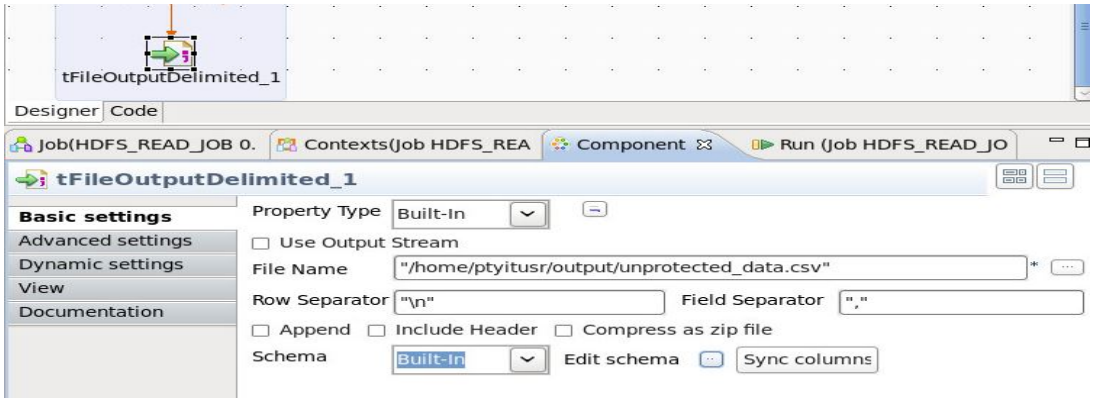


g) Create the *tFileOutputDelimited* component.

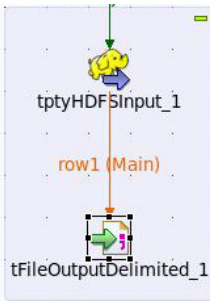
h) Enter the local file system location, where the cleartext data needs to be stored.

i) Enter the *Row Separator* option, as required.

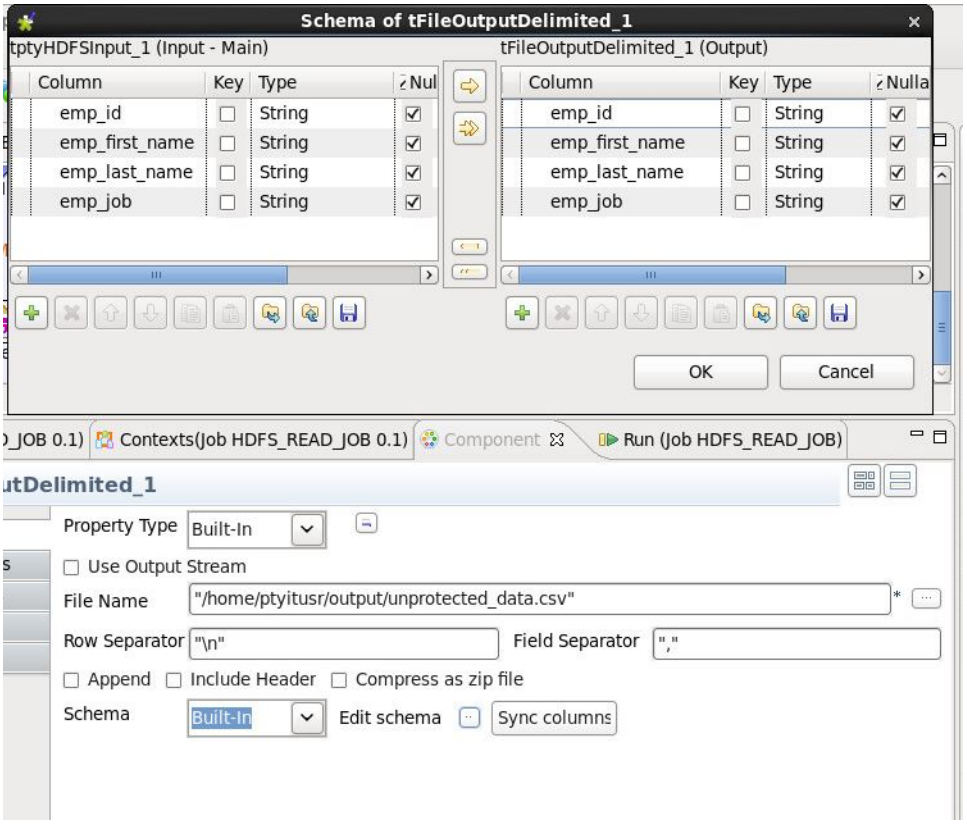
j) Enter the *Field Separator* option, as required.



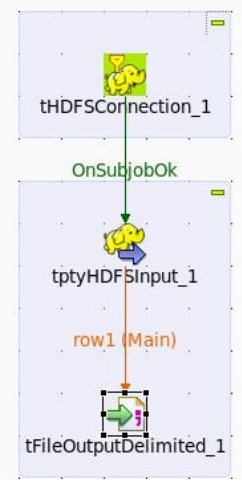
k) Connect the *tptyHDFSInput_1* component to the *tFileOutputDelimited_1* component as illustrated in the following figure.



l) Click **Edit schema** to map the fields in the protected data in the HDFS directory to the local file location, which will store the cleartext data.

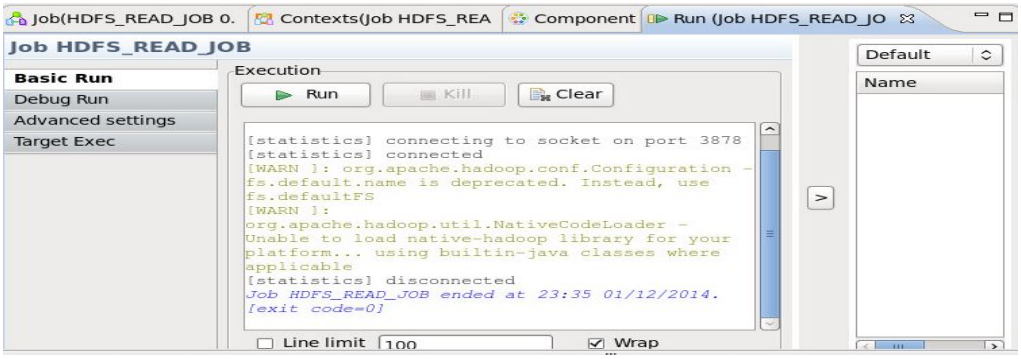


7. Connect all the components in the job diagram, as illustrated by the following figure.



8. Click on the *Run* tab.
   The Run tab appears.

9. Click **Run** to run the job.
   Check the job execution status in the console.



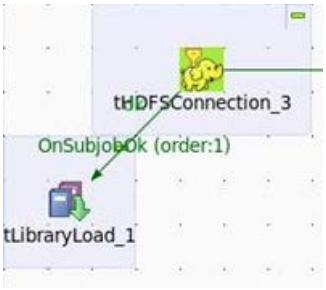
10. Login to the system with the user *ptyitusr* using the following command.
    ```
    >> su - ptyitusr
    ```
11. Navigate to the */home/ptyitusr/output* directory using the following command.
    ```
    >> cd /home/ptyitusr/output
    ```
12. Read the *unprotected_data.csv* file from the HDFS directory */home/ptyitusr/output/* using the following command.
    ```
    >> cat unprotected_data.csv
    ```
    The cleartext data appears.

```
[root@cdh45-1 ~]# su - ptyitusr
-sh-4.1$ cd output/
-sh-4.1$ pwd
/home/ptyitusr/output
-sh-4.1$ ll
total 4
-rw-r--r-- 1 root root 373 Dec  1 23:35 unprotected_data.csv
-sh-4.1$ cat unprotected_data.csv
101,Adam,Wiley,Sales
102,Brian,Chester,Service
103,Julian,Cross,Sales
104,Dylan,Moore,Marketing
105,Chris,Murphy,Service
106,Brian,Collingwood,Service
107,Michael,Muster,Marketing
108,Miley,Rhodes,Sales
109,Chris,Coughlan,Sales
110,Aaron,King,Marketng
111,Adam,Young,Service
112,Tyler,White,Sales
113,Martin,Reeves,Service
114,Michael,Morton,Sales
115,Tim,Rhodes,Marketing
```

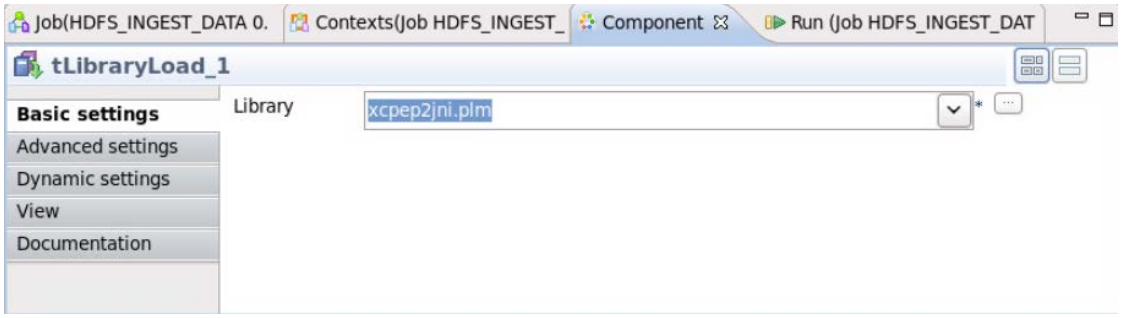
15.8 Configuring Talend Jobs to run with HDFSFP with Target Exec as Remote

➤ **To configure Talend to run jobs with HDFSFP remotely:**

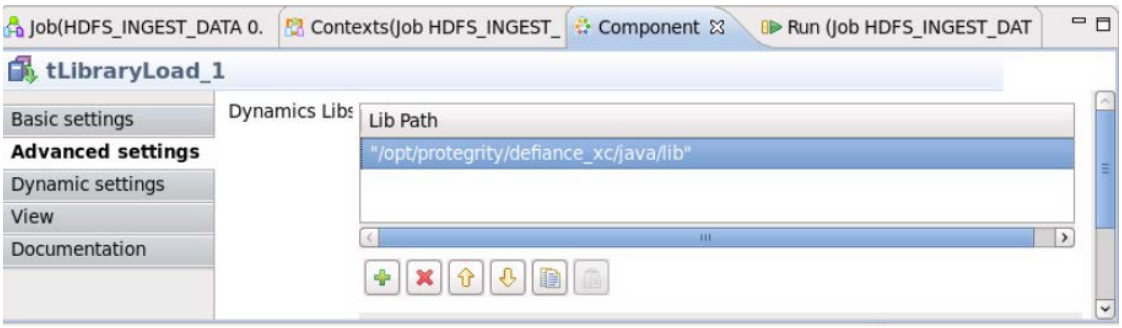
1. Connect the *tLibraryLoad* component in the job diagram, as illustrated by the following figure.



2. In the **Basic settings** tab for the *tLibraryLoad* component, load the *xcpep2jni.plm* file, as illustrated by the following figure.

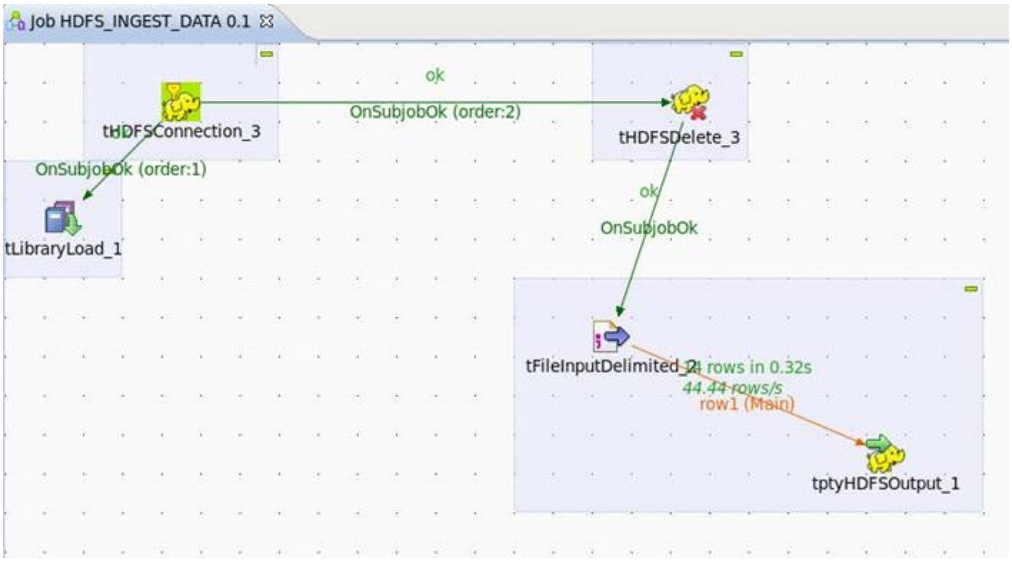


3. In the **Advanced settings** tab for the *tLibraryLoad* component, set the dynamic library path to */opt/protegrity/defiance_xc/java/lib*, as illustrated by the following figure.

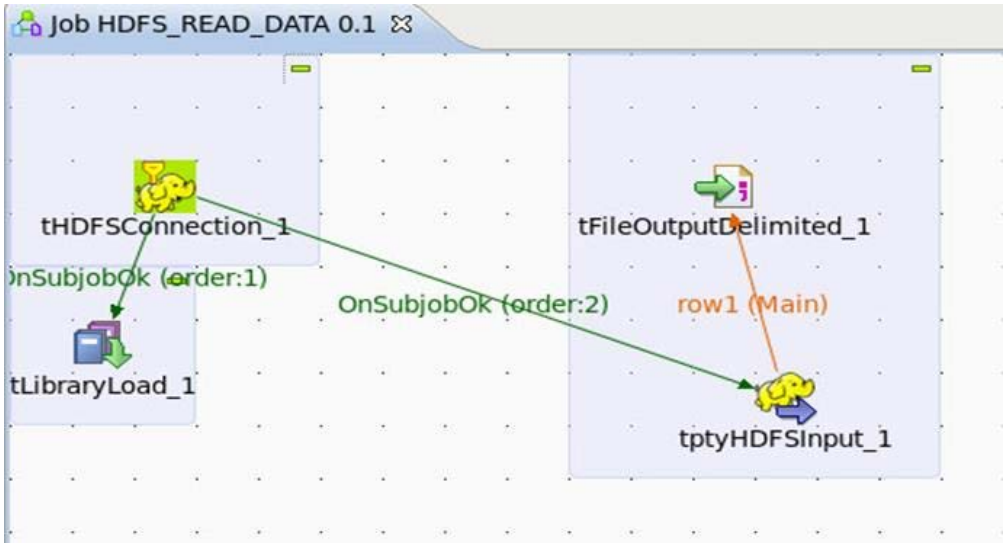


4. Copy the *beuler.properties* file from the *<TALEND_Installation_DIR>/hdfsfp* directory to the home directory of the user, which is being used to run the Talend job server.
5. Ensure that the setting for *Target Exec* is set to *Remote server*, as illustrated by the following figure.

6. If you are performing a *protect* operation, then connect all the components in the job diagram, as illustrated by the following figure.



7. If you are performing an *unprotect* operation, then connect all the components in the job diagram, as illustrated by the following figure.



15.9 Using Talend with HDFSFP and MapReduce

If you need to use Talked with HDFSFP and MapReduce, then perform the following action.

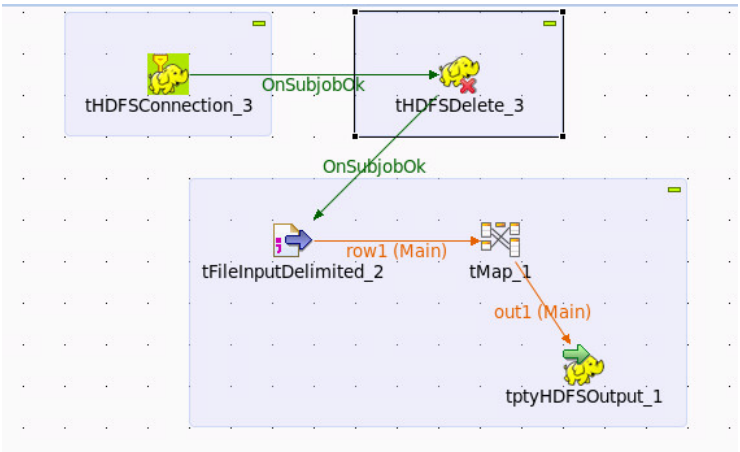
1. Create a Routine in Talend Studio, which contains the Java code for invoking the MapReduce Protector APIs for Protect or Unprotect.
2. Utilize the *tMap* component from the palette to perform the operations for processing the data.

15.9.1 Protecting Data Using Talend with HDFSFP and MapReduce

➤ **To protect data using Talend with HDFSFP and MapReduce:**

1. Access the *.csv* file from the local directory which contains cleartext data.
2. Protect the fields in the *.csv* file using the required token elements.

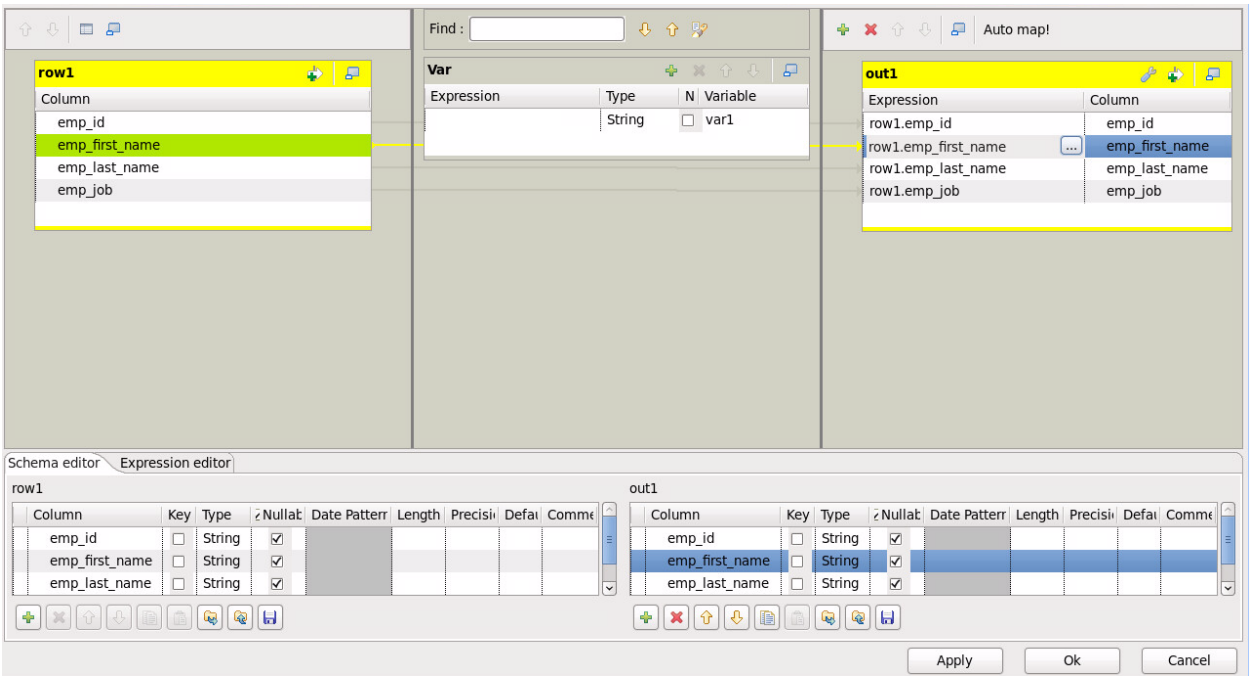
3. To ingest the protected data into the protected HDFS directory, where HDFSFP encryption takes place, connect all components in the job diagram, as illustrated by the following figure.



The routine and the required properties to protect the data reside in the *tMap* component, before it is loaded into an HDFSFP protected table.

4. To view the contents of the *tMap* component, double-click on *tMap*.

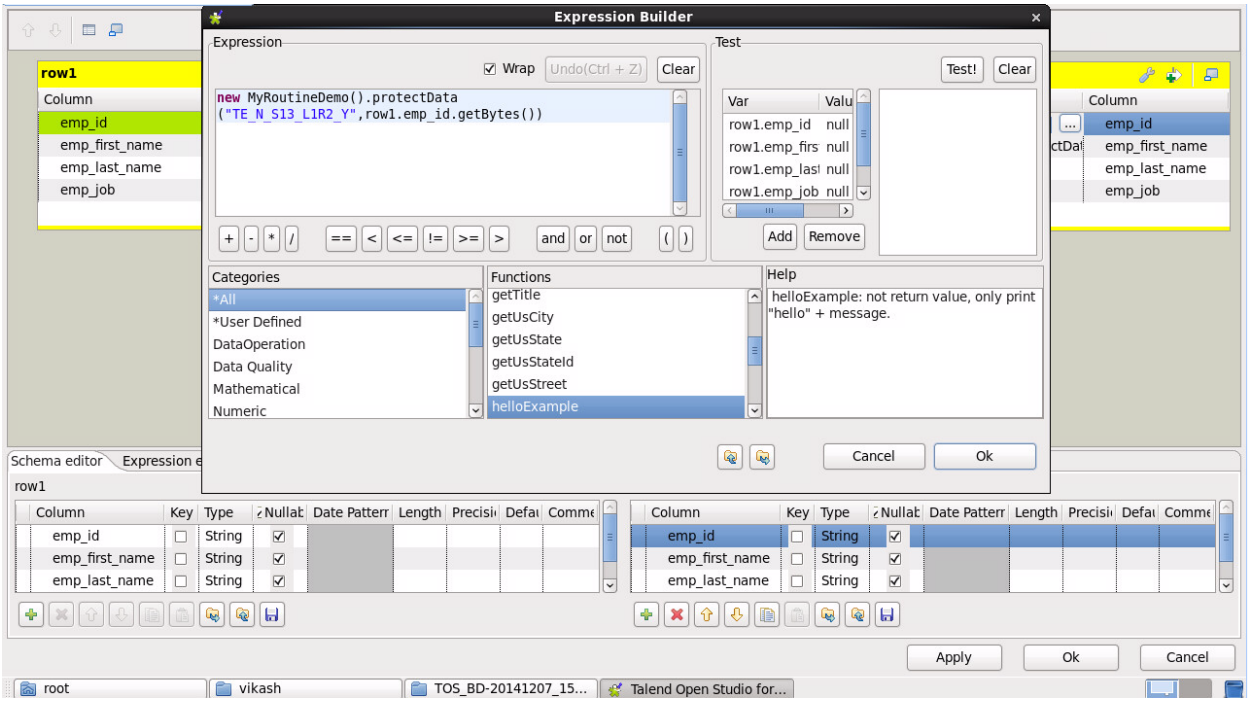
The following screen listing the contents of the *tMap* component appears.



The left pane lists the table with four columns and the right pane lists the routines called for protecting the data.

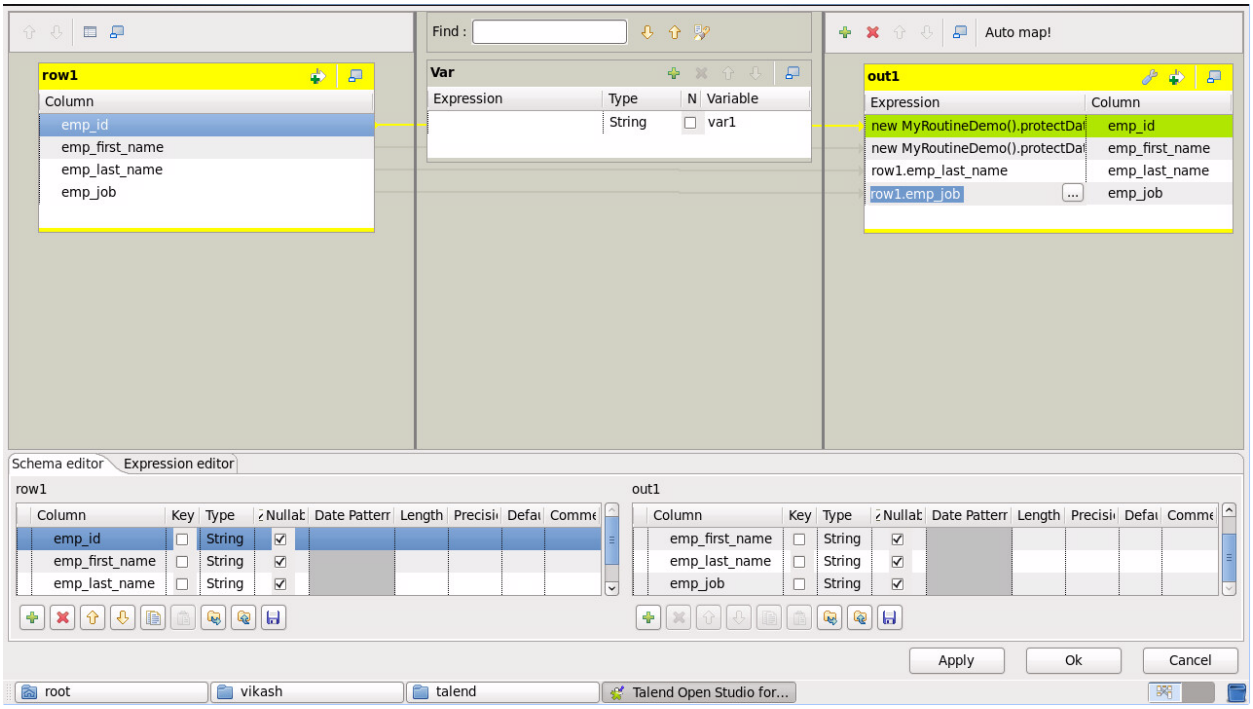
5. To protect the data in the first two rows of the table, click the button beside the rows in the right pane, and as illustrated in the following figure, call the class and functions from the routine using the following syntax,

   new class_name().function name("<Data Element>",<row_no.>.<row.name>)



In the example, the first row and second rows are protected using the function *protectData()* and the token elements *TE_N_S13_L1R2_Y* and *TE_LASCII_L2R1_Y* respectively.

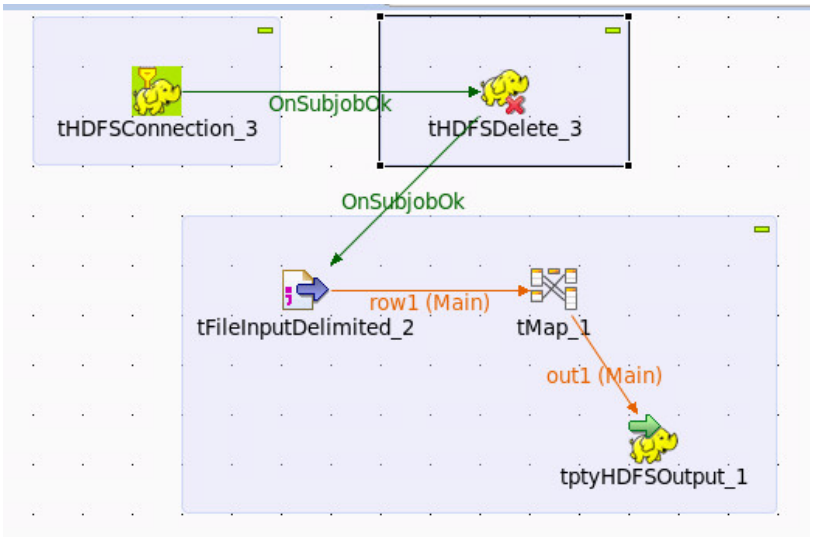
6. After the routine is called for the rows which need to be protected, the right pane illustrates how the *protect* function is called from the Routine.



15.9.2 Unprotecting Data Using Talend with HDFSFP and MapReduce

➤ **To unprotect data using Talend with HDFSFP and MapReduce:**

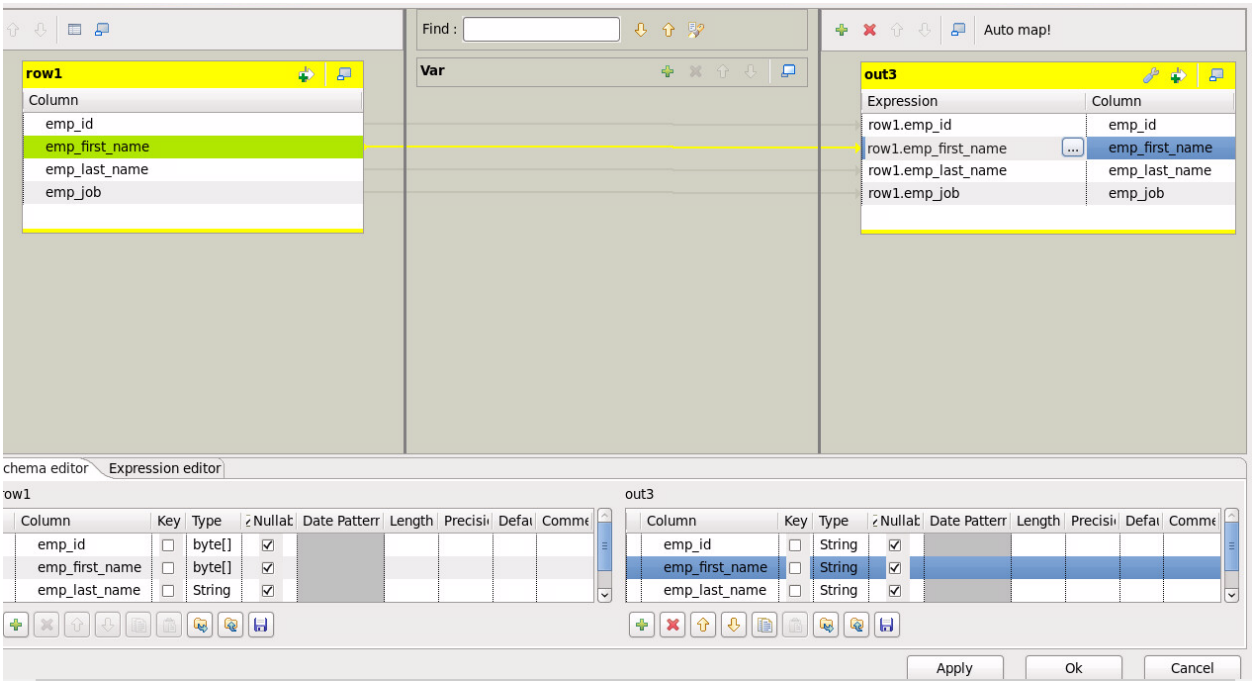
1. Access the protected data from the protected HDFS directory, where HDFSFP decryption takes place.
2. Unprotect the fields in the *.csv* file using the required token elements.
3. To ingest the unprotected data into an output *.csv* file in the local directory, connect all components in the job diagram, as illustrated by the following figure.



The routine and the required properties to protect the data reside in the *tMap* component, before it is loaded into an HDFSFP protected table.

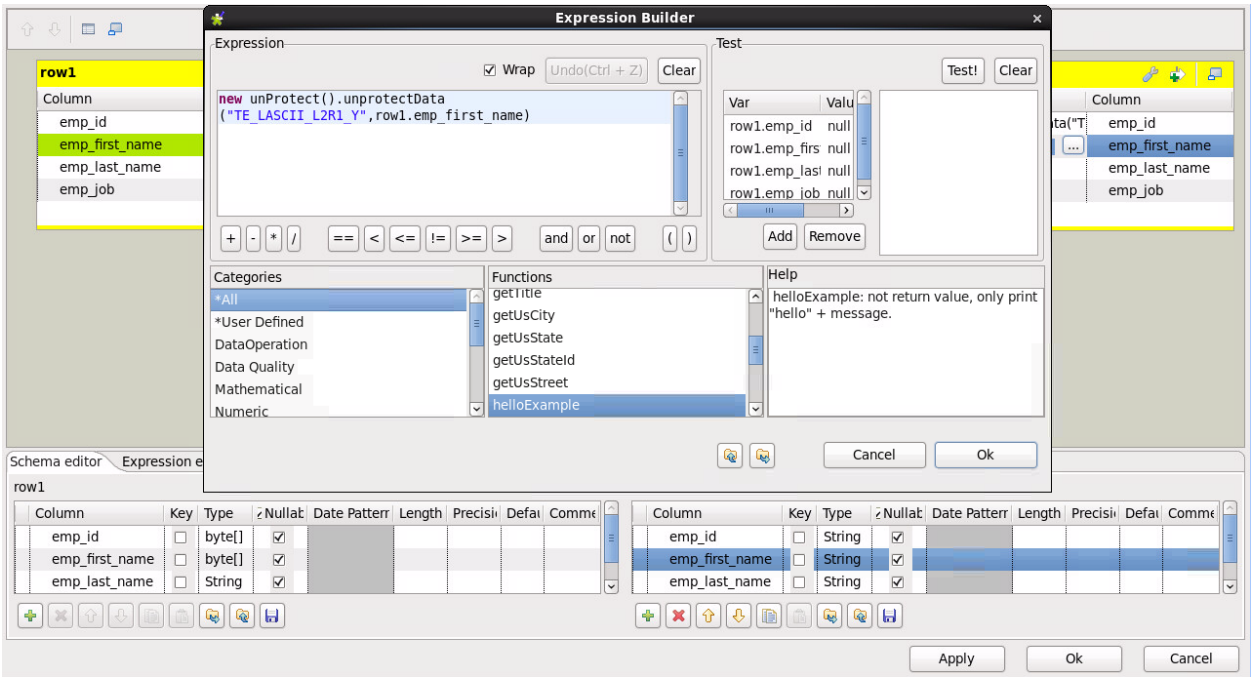
4. To view the contents of the *tMap* component, double-click on *tMap*.

The following screen listing the contents of the *tMap* component appears.



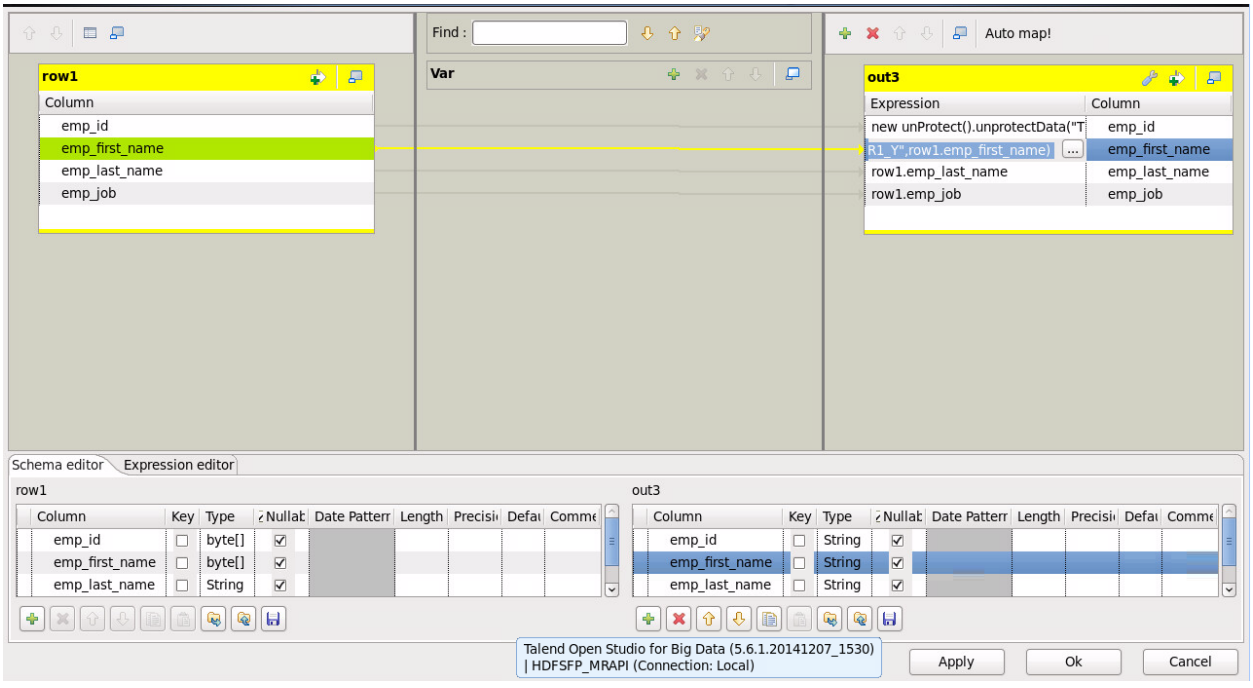
The left pane lists the table with four columns and the right pane lists the routines called for unprotecting the data.

5. To unprotect the data in the first two rows of the table, click the button beside the rows in the right pane, and as illustrated in the following figure, call the class and functions from the routine using the following syntax,
   new class_name().function name("<Data Element>",<row_no.>.<row.name>)



In the example, the first row and second rows are unprotected using the function *unprotectData()* and the token elements *TE_N_S13_L1R2_Y* and *TE_LASCII_L2R1_Y* respectively.

6. After the routine is called for the rows which need to be unprotected, the right pane illustrates how the *unprotect* function is called from the Routine.



15.9.3 Sample Code Usage

The MapReduce sample routine, described in this section, is an example on how to use the Protegrity MapReduce protector APIs. The sample program utilizes the following two routine files:

- **Routine Item** – The main routine calls the Mapper job.
- **Routine Properties** – The properties related to the main routine.

15.9.3.1 Routine Item

```
package routines;

import com.protegrity.hadoop.fileprotector.fs.ProtectorException;
import com.protegrity.hadoop.fileprotector.fs.PtyHdfsProtector;
import com.protegrity.hadoop.mapreduce.ptyMapReduceProtector;

public class MyRoutineDemo {

    public static void helloExample(String message) {
        if (message == null) {
            message = "World"; //$NON-NLS-1$
        }
        System.out.println("Hello " + message + " !"); //$NON-NLS-1$ //$NON-NLS-2$
    }

       public static PtyHdfsProtector protector = new PtyHdfsProtector();

       public void copyFromLocalTest(String[] srcs, String dstf)
       {
       boolean result;
       try {
       result = protector.copyFromLocal(srcs, dstf);
       } catch (ProtectorException pe) {
       pe.printStackTrace();
       }
       }
       public void copyToLocalTest(String srcs, String dstf)
       {
       boolean result;
       try {
       result = protector.copyToLocal(srcs, dstf);
       } catch (ProtectorException pe) {
       pe.printStackTrace();
       }
       }
       public void copyTest(String srcs, String dstf)
       {
       boolean result;
       try {
       result = protector.copy(srcs, dstf);
       } catch (ProtectorException pe) {
       pe.printStackTrace();
       }
       }
       public void mkdirTest(String dir)
       {
       boolean result;
       try {
       result = protector.mkdir(dir);
       } catch (ProtectorException pe) {
       pe.printStackTrace();
       }
       }
       public void moveTest(String srcs, String dstf)
       {
```

```
boolean result;
try {
result = protector.move(srcs,dstf);
} catch (ProtectorException pe) {
pe.printStackTrace();
}
}

public void deleteFileTest(String file,boolean skipTrash)
{
boolean result;
try {
result = protector.deleteFile(file, skipTrash);
} catch (ProtectorException pe) {
pe.printStackTrace();
}
}
public void deleteDirTest(String dir,boolean skipTrash)
{
boolean result;
try {
result = protector.deleteDir(dir, skipTrash);
} catch (ProtectorException pe) {
pe.printStackTrace();
}
}

public String  protectData(String dataElement,byte[] data){
       ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
       mapReduceProtector.openSession("0");
       System.out.println(dataElement);
       System.out.println(new String(data));
       byte[] output = mapReduceProtector.protect(dataElement, data);
       return new String(output);
}

public String  unprotectData(String dataElement,byte[] data){
       ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
       mapReduceProtector.openSession("0");
       System.out.println(dataElement);
       System.out.println(new String(data));
       byte[] output = mapReduceProtector.unprotect(dataElement, data);
       return new String(output);
}

public static void main(String[] args) {
MyRoutineDemo protect = new MyRoutineDemo();
// Ingest Local Data into HDFS
String srcsCFL[] = new String[2];
srcsCFL[0] ="/home/ptyitusr/input/sample.csv";
srcsCFL[1] ="/home/ptyitusr/input/test.csv";
String dstfCFL ="/user/ptyitusr/talend";
protect.copyFromLocalTest(srcsCFL, dstfCFL);

// Extract HDFS file to Local
String srcsCTL= "/user/ptyitusr/talend/prot.csv";
String dstfCTL = "/home/ptyitusr/input";
protect.copyToLocalTest(srcsCTL, dstfCTL);
// Copy File from HDFS to HDFS
String srcsCopy="/user/ptyitusr/talend/prot.csv";
String dstfCopy ="/user/ptyitusr/talend";
protect.copyTest(srcsCopy, dstfCopy);
// Create HDFS Sub-Directory
String dir = "/user/ptyitusr/talend/sub";
protect.mkdirTest(dir);
// Move from HDFS to HDFS
```

```
String srcsMove = "/user/ptyitusr/talend/prot.csv";
String dstfMove = "/user/ptyitusr/talend";
protect.moveTest(srcsMove, dstfMove);
// Delete File from HDFS
String fileDelete = "/user/ptyitusr/talend/prot.csv";
boolean skipTrashFile = false;
protect.deleteFileTest(fileDelete,skipTrashFile);
// Delete Sub-Directory and Children from HDFS
String dirDelete = "/user/ptyitusr/talend";
boolean skipTrashDir = false;
protect.deleteDirTest(dirDelete,skipTrashDir);
}
}
```

15.9.3.2 Routine Properties

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:TalendProperties="http://www.talend.org/properties">
  <TalendProperties:Property xmi:id="_DTUMYDIkEeW6nfU7n79eDQ" id="_DTS-QDIkEeW6nfU7n79eDQ"
label="MyRoutineDemo" creationDate="2015-07-24T11:50:12.646-0500" modificationDate="2016-01-
19T02:06:24.583-0500" version="0.1" statusCode="" item="_DTUMYjIkEeW6nfU7n79eDQ"
maxInformationLevel="WARN" displayName="MyRoutineDemo">
    <author href="../../talend.project#_5-CdsDHkEeWZEZzbK6p_uA"/>
    <informations xmi:id="_-Y8DoDJQEeW6nfU7n79eDQ" level="WARN" text="The local variable result is
never read"/>
    <informations xmi:id="_-Y8DoTJQEeW6nfU7n79eDQ" level="WARN" text="The local variable result is
never read"/>
    <informations xmi:id="_-Y8DojJQEeW6nfU7n79eDQ" level="WARN" text="The local variable result is
never read"/>
    <informations xmi:id="_-Y8DozJQEeW6nfU7n79eDQ" level="WARN" text="The local variable result is
never read"/>
    <informations xmi:id="_-Y9RwDJQEeW6nfU7n79eDQ" level="WARN" text="The local variable result is
never read"/>
    <informations xmi:id="_-Y9RwTJQEeW6nfU7n79eDQ" level="WARN" text="The local variable result is
never read"/>
    <informations xmi:id="_-Y9RwjJQEeW6nfU7n79eDQ" level="WARN" text="The local variable result is
never read"/>
  </TalendProperties:Property>
  <TalendProperties:ItemState xmi:id="_DTUMYTIkEeW6nfU7n79eDQ" path=""/>
  <TalendProperties:RoutineItem xmi:id="_DTUMYjIkEeW6nfU7n79eDQ" property="_DTUMYDIkEeW6nfU7n79eDQ"
state="_DTUMYTIkEeW6nfU7n79eDQ">
    <content href="MyRoutineDemo_0.1.item#/"/>
    <imports xmi:id="_uWRN4DInEeW6nfU7n79eDQ" mESSAGE="" mODULE="hdfsfp.jar" nAME="MyRoutineDemo"
rEQUIRED="true" urlPath="/TalendInstall/Talend-5.6.1/hdfsfp/hdfsfp.jar"/>
    <imports xmi:id="_NMUX8L30EeWz-4GwVQX7Ig" mESSAGE="" mODULE="pepmapreduce-2.7.1.jar"
nAME="MyRoutineDemo" rEQUIRED="true" urlPath="/opt/protegrity/hadoop_protector/lib/pepmapreduce-
2.7.1.jar"/>
  </TalendProperties:RoutineItem>
</xmi:XMI>
```

16 Appendix: Migrating Tokenized Unicode Data from and to a Teradata Database

This section describes the procedures for migrating tokenized Unicode data from and to a Teradata database.

16.1 Migrating Tokenized Unicode Data from a Teradata Database

This section describes the task to unprotect the tokenized Unicode data in Hive, HAWQ, Impala, or Spark, which was tokenized in the Teradata database using the Protegrity Database Protector and then migrated to Hive, HAWQ, Impala, MapReduce, or Spark.

Ensure that the data elements used in the data security policy, deployed on the Teradata Database Protector and Big Data Protector machines are uniform.

➤ **To migrate Tokenized Unicode data from Teradata database to Hive, HAWQ, or Impala and unprotect it using Hive, HAWQ, or Impala protector:**

1. Tokenize the Unicode data in the Teradata database using Protegrity Database Protector.
2. Migrate the tokenized Unicode data from the Teradata database to Hive, HAWQ, or Impala.
3. To unprotect the tokenized Unicode data on Hive, HAWQ, or Impala, ensure that the following UDFs are used, as required:
   - Hive: *ptyUnprotectUnicode()*
   - HAWQ: *pty_UnicodeVarcharSel()*
   - Impala: *pty_UnicodeStringSel()*

➤ **To migrate Tokenized Unicode data from a Teradata database to Hadoop and unprotect it using MapReduce or Spark protector:**

1. Migrate the tokenized Unicode data to the Hadoop ecosystem using any data migration utilities.
2. To unprotect the tokenized Unicode data using MapReduce or Spark, ensure that the following APIs are used, as required:
   - MapReduce: *public byte[] unprotect(String dataElement, byte[] data)*
   - Spark: *void unprotect(String dataElement, List<Integer> errorIndex, byte[][] input, byte[][] output)*
3. Convert the protected tokens to bytes using UTF-8 encoding.
4. Send the data as input to the Unprotect API in the MapReduce or Spark protector, as required.
5. Convert the unprotected output in bytes to *String* using UTF-16LE encoding.

   The *string* data will display the data in cleartext format.

The following sample code snippet describes how to unprotect the Tokenized Unicode data, that is migrated from a Teradata database to Hadoop, using the MapReduce or Spark protector.

```
private Protector protector = null;
String[] unprotectinput= new String[SIZE] ;
byte[][] inputValueByte = new byte [unprotectinput.length][];
StringBuilder unprotectedString = new StringBuilder();
int x=0;
for (x=0; x< unprotectinput.length; x++)
inputValueByte[x]= unprotectinput[x].getBytes(StandardCharsets.UTF_8);  // Point a
implementation
```

```
protector.unprotect(DATAELEMENT_NAME, errorIndexList, inputValueByte, outputValueByte); //
Point b implementation
unprotectedString.apprend(new String(outputValueByte[j],StandardCharsets.UTF_16LE))//Point c
implementation
```

16.2 Migrating Tokenized Unicode Data to a Teradata Database

This section describes the task to protect Unicode data in Hive, HAWQ, Impala, MapReduce, or Spark, migrate it to a Teradata database, and then unprotect the tokenized Unicode data using the Protegrity Database Protector.

Ensure that the data elements used in the data security policy, deployed on the Teradata Database Protector and Big Data Protector machines are uniform.

➤ **To migrate Tokenized Unicode data using Hive, HAWQ, or Impala protector to Teradata database:**

1. To protect the Unicode data on Hive, HAWQ, or Impala, ensure that the following UDFs are used, as required:
   - Hive: *ptyProtectUnicode()*
   - HAWQ: *pty_UnicodeVarcharIns()*
   - Impala: *pty_UnicodeStringIns()*
2. Migrate the tokenized Unicode data from Hive, HAWQ, or Impala to the Teradata database.
3. To unprotect the tokenized Unicode data in the Teradata database, use the Protegrity Database Protector.

➤ **To protect Unicode data using MapReduce or Spark protector and migrate it to a Teradata database :**

1. Convert the cleartext format Unicode data to bytes using UTF-16LE encoding.
2. To migrate the tokenized Unicode data using MapReduce or Spark to the Teradata database, ensure that the following APIs are used, as required:
   - MapReduce: *public byte[] protect(String dataElement, byte[] data)*
   - Spark: *void protect(String dataElement, List<Integer> errorIndex, byte[][] input, byte[][] output)*
3. Send the data as input to the Protect API in the MapReduce or Spark protector, as required.
4. Convert the protected output in bytes to *String* using UTF-8 encoding.
   The output is protected tokenized data.
5. Migrate the protected data to the Teradata database using any data migration utilities.

The following sample code snippet describes how to protect Unicode data using the MapReduce or Spark protector, and migrating it to a Teradata database.

```
private Protector protector = null;
String[] clear_data = new String[SIZE] ;
byte[][] inputValueByte = new byte [clear_data.length][];
StringBuilder protectedString = new StringBuilder();
inputValueByte= data.getBytes(StandardCharsets.UTF_16LE); //Point a implementation
protector.protect(DATAELEMENT_NAME, errorIndexList, inputValueByte, outputValueByte); //
Point b implementation
int x=0;
for (x=0; x<outputValueByte.length; x++)
protectedString.append(new String(outputValueByte[x],StandardCharsets.UTF_8)); //Point c
implementation
```