Protegrity Application Protector Guide 9.1.0.0

Created on: Nov 19, 2024

Notice

Copyright

Table of Contents

Chapter 1

Introduction to this Guide

This Protegrity Application Protector Guide discusses about the Application Protector and its uses. The Guide also provides detailed information and examples of libraries and deployment architectures for all flavors of the Protegrity Application Protector (AP).

The purpose of this guide is to help you understand and implement the APIs in your application. Developers who use this guide should be familiar with the operation and management of Protegrity Data Security Platform and Protegrity Application Protector (AP).

This guide includes AP API libraries and definitions and code samples. *Hello World* examples are provided as well to assist in explaining the required steps and requirements for the Protegrity AP API.

The code samples in this guide represent a small portion of code and are designed to show the basic logic and programming constructions needed to use the Protegrity AP API effectively.

For more information about AP API libraries and definitions, code samples, and error codes, refer to the section *Application Protector* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

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 Protegrity Application Protector (AP) Implementation* introduces you to the concepts of the Application Protector, discusses the configuration of the Protector, and the APIs connected with this Protector.
- *Section 3 Application Protectors C* discusses the AP C having three variants – AP Standard, AP Client, and AP Lite. The deployment architecture, setup details and API list have been included.
- *Section 4 Application Protector Java* provides the architecture, features, and configuration along with a list of all APIs. A sample application explains how the Protector can be used to protect and unprotect.
- *Section 5 Application Protector Python* provides the architecture, features, and configuration along with a list of all APIs. A sample application explains how the Protector can be used to protect, reprotect and unprotect.
- *Section 6 Application Protector Golang* provides the architecture, features, and configuration along with a list of all APIs. A sample application explains how the Protector can be used to protect and unprotect.
- *Section 7 Application Protector NodeJS* provides the architecture, features, and configuration with a list of all APIs. A sample application explains how the Protector can be used to protect and unprotect.
- *Section 8 Application Protector .Net* provides the architecture, features, and configuration with a list of all APIs. A sample application explains how the Protector can be used to protect and unprotect.

- *Section 9 Appendix A: Multi-node Application Protector Architecture* describes the multi-node Application Protector architecture, its individual components, and how logs are collected using the new logging component, Log Forwarder.
- *Section 10 Appendix B: Using Go Module with Private GitLab Repository* describes the steps to use the Go module.

1.2 Accessing the Protegrity documentation suite

This section describes the methods to access the *Protegrity Documentation Suite* using the *My.Protegrity* portal.

1.2.1 Viewing product documentation

The **Product Documentation** section under **Resources** is a repository for Protegrity product documentation. The documentation for the latest product release is displayed first. The documentation is available in the HTML format and can be viewed using your browser. You can also view and download the *.pdf* files of the required product documentation.

1. Log in to the *My.Protegrity* portal.
2. Click **Resources** > **Product Documentation**.
3. Click a product version.
   The documentation appears.

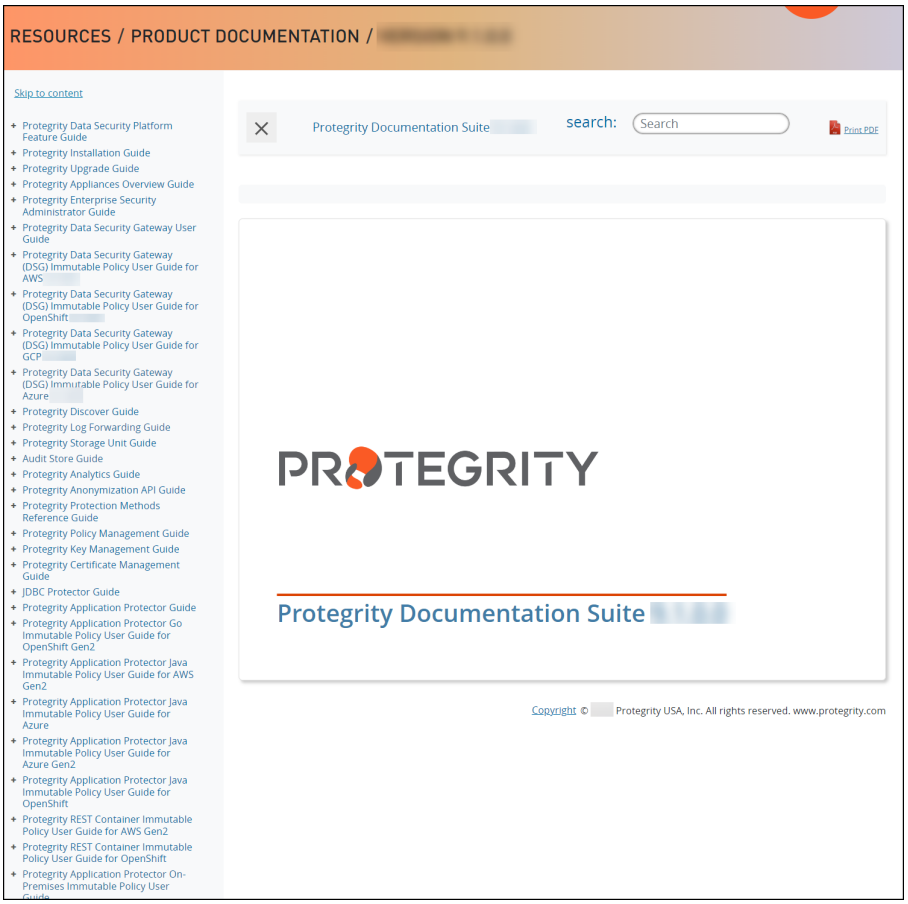

Figure 1-1: Documentation

4. Expand and click the link for the required documentation.
5. If required, then enter text in the **Search** field to search for keywords in the documentation.
   The search is dynamic, and filters results while you type the text.
6. Click the **Print PDF** icon from the upper-right corner of the page.
   The page with links for viewing and downloading the guides appears. You can view and print the guides that you require.

1.2.2 Downloading product documentation

This section explains the procedure to download the product documentation from the *My.Protegrity* portal.

1. Click **Product Management** > **Explore Products**.
2. Select **Product Documentation**.
   The **Explore Products** page is displayed. You can view the product documentation of various Protegrity products as per their releases, containing an overview and other guidelines to use these products at ease.
3. Click **View Products** to advance to the product listing screen.
4. Click the **View** icon ( ) from the **Action** column for the row marked **On-Prem** in the **Target Platform Details** column.
   If you want to filter the list, then use the filters for: **OS**, **Target Platform**, and **Search** fields.
5. Click the icon for the action that you want to perform.

Chapter 2

Protegrity Application Protector (AP) Implementation

2.1 Introduction to Protegrity Application Protector

The Protegrity Application Protector (AP) is a high-performance, versatile solution that provides a packaged interface to integrate comprehensive, granular security and auditing into enterprise applications. It eliminates the need for application developers to master the complexities of cryptography, while keeping the security team in control of sensitive data protection and access.

2.1.1 Features

This section describes the features of the Protegrity Application Protector.

Simple Interface

Based on a simple programming interface that can be accessed from a variety of programming languages, Protegrity AP APIs eliminate the need for application developers to master the complexities of cryptography.

Separation of Duties

The security regulations require that there is a clear separation of duties between the Security Administrators and application developers. The concept of separation of duties is typically extended to require that the developers are not the authors of application security policies. As a packaged sub-system, Protegrity AP enforces this separation of responsibilities by encapsulating all aspects of security and providing central control to the Security Administrator with full audit capabilities.

Central Control of Security Policy

The Protegrity Data Security Platform allows the Security Administrator to centrally define corporate data security policies, which are then deployed to protection points throughout the enterprise. Policies for Protegrity AP are administered on the Enterprise Security Administrator (ESA) using the Policy Management.

Protegrity AP is one of the protection points offered by Protegrity, and by integrating Protegrity AP into an application, the Protegrity Data Security Platform will control the enforcement, auditing, and reporting functions of Protegrity AP.

Policy-based Access Control

Protegrity AP enforces all aspects of a security policy, including role and user-based rights and limitations. The Security Officer is responsible for setting the protection/unprotection functions and the protection options for the data elements.

Each call to the Protegrity AP identifies the calling application and/or the authorized application user. Protegrity AP automatically validates the calling party against the roles and users defined in the corporate security policy.

It also determines the user access rights to data, the right to encrypt, decrypt, and re-encrypt the data.

In addition to granting access rights, a security policy defines the level of audit trail and record keeping for particular applications or users. It delivers fine grained auditing that can log all access, only encryption or decryption access, tokenization access, or provide no logging at all.

Without any additional development by the application developer, Protegrity AP enforces corporate access and logging policy.

Centralized Key Management

The most important element of any encryption strategy is encryption key management. Keys must be protected at all times, and must be secured for disaster recovery. The Protegrity Data Security Platform deploys patented key management technology that securely manages all aspects of encryption key operations in the Protegrity AP services.

Since keys are centrally managed, information can flow throughout the enterprise in a continuously secure manner, and keys can be shared across computing environments to enable data that is encrypted at one location, to be decrypted and used at a different location.

A typical example is to enable point of sale (POS) information to be immediately encrypted by a call to Protegrity AP at the remote store location, and then securely transmitted back to the corporate headquarters for use in other applications or stored in databases that are also protected by the Protegrity Data Security Platform.

AP Encryption and Tokenization

The Protegrity AP uses the Protegrity Data Security Platform policy and encryption or tokenization settings providing protection for the data elements. The AP supports the following protection algorithms: 3DES, AES-128, AES-256, CUSP 3DES, CUSP AES-128, CUSP AES-256, Hashing, FPE-FF1, HMAC-SHA1, Token Element, and No Encryption.

2.1.2 Limitations

The external IV is used as a parameter to the protection call for producing different protected data. It is ignored when using encryption data elements.

The float, double, and datetime APIs do not support the external IV.

2.2 Protegrity AP API Implementation

The Protegrity AP APIs are created to make the operations of protecting, unprotecting, and reprotecting data easier with Protegrity AP from an application. A function is also available to calculate the HMAC hash value. The APIs are used on the supported programming languages.

The data security activities are carried out by the Application Protector using the PEP server.

The Protegrity AP APIs can be used with different implementations. The AP APIs are implemented for the following types of Application Protectors:

AP C provides APIs that integrate with the customer application to protect, unprotect, and reprotect sensitive data. The AP C can be used with any customer application that is developed using the C programming language.

AP Java provides APIs that integrate with the customer application to protect, unprotect, and reprotect sensitive data. The AP Java can be used with any customer application that is developed using the Java programming language.

AP Python provides APIs that integrate with the customer application to protect, unprotect, and reprotect sensitive data. The AP Python can be used with any customer application that is developed using the Python programming language.

AP Golang provides APIs that integrate with the customer application to protect, unprotect, and reprotect sensitive data. The AP Golang can be used with any customer application that is developed using the Golang programming language.

AP NodeJS provides APIs that integrate with the customer application to protect, unprotect, and reprotect sensitive data. The AP NodeJS can be used with any customer application that is developed for NodeJS.

Chapter 3

Application Protector C

The Protegrity Application Protector (AP) C provides APIs that integrate with the customer application to protect, unprotect, and reprotect sensitive data. The AP C can be used with any customer application that is developed using the C programming language.

> **Note:**
> The AP C Protector has been updated in the build 9.1.0.0.x and this version of the document applies to the AP C Protector 9.1.0.0.x and higher.
>
> If you are using an earlier version of the AP C Protector, then refer to the document included in the documentation package with the build prior to 9.1.0.0.x.

The AP C has the following protection and security access methods:

- Get product version
- Check access rights for the user
- Get current key id
- Get key id from protected data
- Get default data element
- Protect
- Unprotect
- Reprotect

For more information about the AP C protection and security access methods, refer to the section *Application Protector (AP) C APIs* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.0*.

3.1 Features

This section describes the features of the Application Protector C.

Supported C Distributions

The AP C supports GCC versions 3.2.3 and higher for all operating systems.

Session Validity

A session is valid until the *XCTerminateLib* API is called in the application.

Audit Logs

The AP C Session API checks sessions to handle the generation of audit records and provide information on the error in the preceding or last session. Each session generates an audit record for every new protection method call combined with the data element used.

For single data item calls, a total of three audit log events are generated if you perform the following operations:

- One protect operation with data element name *a* (with count as 1)
- Five protect operations with data element *b* (with count as 5)
- One thousand unprotect operations with data element *a* (with count as 1000)

The same behavior is seen with Bulk data items, every data protection operation, such as, protect, unprotect, and reprotect will generate audit log events. For example, if the bulk size is 3 and a total of two bulk protect operations, having same data elements and session, are performed then one audit log will be generated with count as 6.

You can use this knowledge to ensure how you want the audit records to be generated, in case of single or bulk data item calls, by deciding how long a session is valid and how often new sessions are created using the *XCOpenSession* API.

> **Caution:**
> When a short running application has completed its execution (in less than a second), the audit logs will not be seen. In such cases, the *XCFlushPepAudits* function() API needs to be invoked.
>
> For more information about the *XCFlushPepAudits* function, refer to the section *XCFlushPepAudits Function* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

An audit log for the trusted application is displayed each time an application is initialized, indicating whether the initialization was successful or not. The audits are created only after the *xcapi.plm* file are loaded. The audits are generated in the ESA Forensics for easy access by the Security Officer.

3.2 Architecture and Components

This section describes the architecture, the individual components, and the workflow of the Protegrity Application Protector (AP) C solution.

The following figure shows the deployment architecture for the AP C.

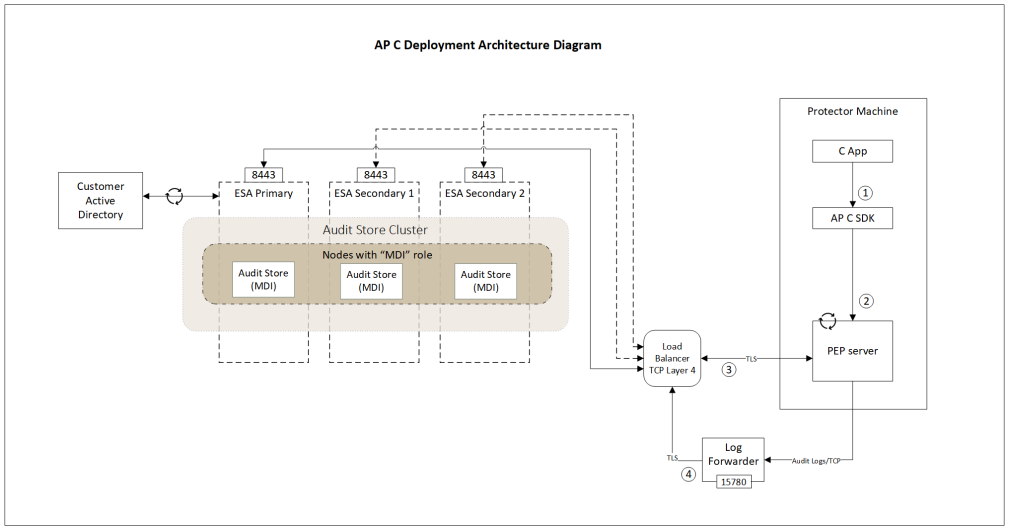

Figure 3-1: AP C Deployment Architecture

The following table describes the components of the AP C deployment architecture.

Table 3-1: Components of AP C Deployment Architecture

Component	Description
C App	Customer C Application
AP C SDK	Protegrity Application Protector C SDK
PEP Server	Component that downloads the policy package from the ESA and reads the package to perform any operations, such as, protect, unprotect, and reprotect
Enterprise Security Administrator (ESA)	Component of the Data Security Platform that consists of the following: • ESA Primary - Manages the policies, keys, monitoring, auditing and reporting of the protected systems in the enterprise • ESA Secondary 1 and 2 - Additional ESAs for Disaster Recovery
Log Forwarder	Log processing tool that collects the logs from the protectors and forwards them to the Audit Store Cluster
Load Balancer	Component that efficiently distributes the calls from the protector to the ESA
Customer Active Directory	Customer data that synchronizes with the ESA

The following steps describe the workflow of a sample AP C deployment in the production environment.

1. The customer C application initializes the Protegrity AP C SDK.
2. During the initialization process, the PEP server synchronizes with the ESA to download the policy over a TLS channel and stores it in the shared memory.
3. The protector performs the data security operations, such as, protect, unprotect, or reprotect, using the downloaded policy.
4. The audit logs generated while performing the operations are forwarded to the ESA using the Log Forwarder.

3.3 Working with the PEP Server

The Policy Enforcement Point (PEP) server is a component that downloads the policy package from the ESA and reads the package to perform any operations, such as, protect, unprotect, and reprotect. It is a communication agent that accepts the policy deployed from the Hub Controller.

The Policy Enforcement Agent is responsible for the actual enforcement of the policy and protection of sensitive data. Its implementation differs based on the host technology what the protector protects - application, database, or file. The ESA allows security officers to easily specify data security requirements and distribute them across the enterprise to be executed locally.

Once the policy is determined and set in the ESA, it is deployed to Protegrity Data Protectors for enforcement on their installed systems, ensuring consistent security enterprise-wide.

For more information about the various configurations of the PEP server using the *pepserver.cfg* file, refer to the section *Appendix: PEP Server Configuration File* in the *Installation Guide 9.1.0.4*.

3.4 Working with the Log Forwarder

The Log Forwarder component is a log processing tool that collects the data security operation logs from the Application Protector and forwards them to the Audit Store on the ESA.

The Log Forwarder uses ports, such as *15780*, which is configurable to transport protection and audit logs to the ESA. The ESA receives the logs and stores it in the Audit Store.

For more information about the Log Forwarder, refer to the section *Components of the Logging Architecture* in the *Protegrity Log Forwarding Guide 9.1.0.1*.

3.5 Running AP C - Example

This section describes how to use the AP C protect, unprotect, and reprotect APIs. It is assumed that the ESA is already available, and the AP C has been successfully installed.

The tasks can be divided in the following order.

1. Create the data elements and data store in the Policy Management on the ESA Web UI.
2. Create the member sources and roles.
3. Configure the policy.
4. Configure the trusted application.
5. Add a trusted application to the data store.
6. Install the AP C.
7. Use the protection and unprotection APIs by first protecting the string array and then unprotecting the protected value.

For more information on running the AP C sample application, refer to the section *Running IAP - Example*.

3.6 BulkProtect Function in C Sharp

This section describes how to configure and use the AP C with the C Sharp sample code.

> **Note:**
> C Sharp is not a part of the Protegrity Application Protector Suite. This section is provided to assist the configuration of the AP C with C Sharp.

A C Sharp sample code, *BulkProtect*, has been provided to call the bulk protect and unprotect APIs provided by the AP C. The sample code is written to perform Alpha-Numeric tokenization using a sample user *exampleuser1* and a sample data element *alphanum*. In the sample code, a hardcoded text string *teststring1234* is passed twice as input to simulate bulk data. The *BulkProtect* sample code first protects this bulk data, and then unprotects it.

This section provides the following details:

- *Prerequisites for Running the BulkProtect Sample Code*

- *Running the BulkProtect Sample Code*
- *Walkthrough of the BulkProtect Sample Code*

3.6.1 Prerequisites for Running the BulkProtect Sample Code

This section describes the prerequisites for running the *BulkProtect* sample code.

Ensure that the following prerequisites are met before running the *BulkProtect* sample code:

- Setup the Application Protector C on a Windows platform. The *BulkProtect* sample code is included in the `\Protegrity\Defiance XC\samples\xccsharpsample` directory. In addition, start the PEP server service for the ESA.

  For more information about setting up the Application Protector C on a Windows platform, refer to the section *Setting up Application Protectors C on Windows* in the *Installation Guide 9.1.0.4*.

- Setup the Visual Studio environment and compile the *BulkProtect* sample code into the *BulkProtectSample.exe* file.

  > **Note:**
  > For more information about compiling the sample code, refer to the *readme.txt* file available in the *xccsharpsample* directory.

- Copy the *xcpep.plm* library from the `\Protegrity\Defiance XC\bin` directory to the workspace of the *BulkProtect* sample code.
- Deploy the required data policy. The data policy must contain the sample user *exampleuser1* and the sample data element *alphanum*. The sample user must have protect and unprotect permissions.

  For more information about deploying policies, refer to the section *Creating and Deploying Policies* in the *Policy Management Guide 9.1.0.4*.

3.6.2 Running the BulkProtect Sample Code

This section describes how to run the *BulkProtect* sample code.

To run the *BulkProtect* sample code:

1. Start the command prompt from the Windows machine in the administrator mode.
2. Change the current working directory path to the location where the *BulkProtectSample.exe* file is placed.
3. Run the *BulkProtectSample.exe* executable file.
   The *BulkProtect* sample code first protects the sample data and then unprotects it. The results of the bulk protect and bulk unprotect APIs are displayed on the console.

```
c:\BulkProtectSample>bulkprotectsample.exe
*********************************************************
* Protegrity C# sample Implementation for Bulk operations *
*********************************************************
Application Protector version 9.1.0.0.4
User         : exampleuserl
Data Element : alphanum


PROTECTING EXAMPLE DATA
===============================================
Input               Output          Error Code
===============================================
teststringl234    FgvH5CjnHYkm8Y       6
teststringl234    FgvH5CjnHYkm8Y       6
```

```
UNPROTECTING PROTECTED EXAMPLE DATA
============================================
Input              Output          Error Code
============================================
FgvH5CjnHYkm8Y    teststringl234       8
FgvH5CjnHYkm8Y    teststringl234       8
```

For information on how the BulkProtect sample code is executed, refer to the section *Walkthrough of the BulkProtect Sample Code*.

3.6.3 Walkthrough of the BulkProtect Sample Code

This section provides a walkthrough of the *BulkProtect* sample code and describes the individual code snippets that are a part of the sample code.

When you run the *BulkProtectSample.exe* file, the sample code internally executes the following major steps:

1. *Retrieve the Application Protector version.*
2. *Initialize libraries.*
3. *Open a session.*
4. *Perform bulk protection.*
5. *Perform bulk unprotection.*
6. *Close the session.*
7. *Terminate libraries.*

3.6.3.1 Retrieving the Application Protector Version

This section provides a sample code to retrieve the version number of the installed AP C and display it on the console.

```
/* Retrieving XCAPI version */
      returnCode = xcapi.GetVersion( outputBuffer, ( int )outputBufferLength );
      if( returnCode != xcdefinitions.XC_SUCCESS )
      {
        throw new Exception( "Failed to get application protector library version" );
      }

      xcVersion = ASCIIEncoding.ASCII.GetString( outputBuffer, 0, 32 ).TrimEnd( '\0' );
      Console.WriteLine( "Application Protector version " + xcVersion );
```

3.6.3.2 Initializing Libraries

This section provides a sample code to initialize the Application Protector libraries.

The *xcpep.plm* file is used for the libraries that are imported in the *xcapi.cs* file. All the return codes are defined in the *xcdefinitions.cs* file.

For more information about the AP C return codes, refer to the section *Application Protectors API Return Codes* in the *Protegrity Troubleshooting Guide 9.1.0.3*.

```
/* Initializing libraries */
      returnCode = xcapi.InitiateLibrary( ref handle, parameter );
      if( returnCode != xcdefinitions.XC_SUCCESS )
        throw new Exception( "Failed to initialize library" );

      parameter = communicationId.ToString();
```

3.6.3.3 Opening a Session

This section provides a sample code for opening a session and returning a handle for that session. This handle is then used in the bulk protect and unprotect APIs. This sample code also displays the sample user and data element on the console.

```
/* Opening session for further operations */
      returnCode = xcapi.OpenSession( handle, "", "", parameter, ref session );
      if( returnCode != xcdefinitions.XC_SUCCESS )
      {
        throw new Exception( "Failed to open session" );
      }

      Console.WriteLine( "User        : " + policyMember );
      Console.WriteLine( "Data Element: " + dataElementName );
```

3.6.3.4 Performing Bulk Protection

This section provides a sample code that can be used to perform bulk protect operations. It displays the input data and the output protected data on the console.

```
/* BulkProtection */
        returnCode = xcapi.BulkProtect( handle,
                                        session,
                                        xcdefinitions.XC_EVENT_FIRST_CALL,
                                        policyMember,
                                        dataElementName,
                                        pcExternalIV,
                                        ui4ExternalIVLength,
                                        inputItems.Ptr,
                                        inputItems.Count,
                                        outputItems.Ptr,
                                        ref itemsCount,
                                        ref errorIndex,
                                        xcParamsEx.Ptr,
                                        xcParamsEx.Size,
                                        pstActionResult.Ptr );


        /*  checking error code of bulk operations */
        if( returnCode != xcdefinitions.XC_SUCCESS )
        {
          try
          {
            BulkProtectSample.outputErrorDescription( handle, session, "Failed to protect
input data",
                                                      returnCode );
          }
          catch( SystemException )
          { }
        }

        /* retreving and print output to the console. */
        Console.WriteLine( "" );
        Console.WriteLine( "" );
        Console.WriteLine( "PROTECTING EXAMPLE DATA" );
        Console.WriteLine( "============================================" );
        Console.WriteLine( " Input            Output              Error Code " );
        Console.WriteLine( "============================================" );

        for( i4Index = 0; i4Index < itemsCount; i4Index++ )
          Console.WriteLine( "{0}     {1}       {2}", inputItems[i4Index].Value,
outputItems[i4Index].Value,
                             outputItems[i4Index].ErrorCode );
```

3.6.3.5 Performing Bulk Unprotection

This section provides a sample code that can be used to unprotect the data, which was protected by the *BulkProtect* API. It displays the protected input data and the unprotected output data on the console.

```
/* BulkUnProtection */
        returnCode = xcapi.BulkUnprotect( handle,
                                          session,
                                          xcdefinitions.XC_EVENT_FIRST_CALL,
                                          policyMember,
                                          dataElementName,
                                          pcExternalIV,
                                          ui4ExternalIVLength,
                                          outputItems.Ptr,
                                          outputItems.Count,
                                          DecItems.Ptr,
                                          ref itemsCount,
                                          ref errorIndex,
                                          xcParamsEx.Ptr,
                                          xcParamsEx.Size,
                                          pstActionResult.Ptr );


        if( returnCode != xcdefinitions.XC_SUCCESS )
        {
          BulkProtectSample.outputErrorDescription( handle, session, "Failed to protect input
data",
                                                    returnCode );
        }

        Console.WriteLine( "" );
        Console.WriteLine( "" );
        Console.WriteLine( "UNPROTECTING PROTECTED EXAMPLE DATA" );
        Console.WriteLine( "================================================" );
        Console.WriteLine( " Input            Output              Error Code " );
        Console.WriteLine( "================================================" );

        for( i4Index = 0; i4Index < itemsCount; i4Index++ )
        {
          Console.WriteLine( "{0}      {1}            {2}", outputItems[i4Index].Value,
                             DecItems[i4Index].Value, DecItems[i4Index].ErrorCode );
        }
```

3.6.3.6 Closing the Session

This section provides a sample code to close the session.

```
/* closing session */
      if( ( System.IntPtr )0 != session )
      {
        xcapi.CloseSession( handle, ref session );
      }
```

3.6.3.7 Terminating Libraries

This section provides a sample code to terminate the AP C libraries.

```
/* Terminating libraries */
      if( ( System.IntPtr )0 != handle )
      {
        xcapi.TerminateLibrary( ref handle );
      }
```

Chapter 4

Application Protector Java

The Protegrity Application Protector (AP) Java provides APIs that integrate with the customer application to protect, unprotect, and reprotect sensitive data. The AP Java can be used with any customer application that is developed using the Java programming language.

> **Note:**
> The AP Java Protector has been updated in the build 9.1.0.0.x and this version of the document applies to the AP Java Protector 9.1.0.0.x and higher.
>
> If you are using an earlier version of the AP Java Protector, then refer to the document included in the documentation package with the build prior to 9.1.0.0.x.

The AP Java has the following protection and security access methods:

- Get product version
- Get last error
- Get current key id
- Get key id
- Get default data element
- Protect
- Unprotect
- Reprotect

For more information about the AP Java protection and security access methods, refer to the section *Application Protector (AP) Java APIs* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.0*.

4.1 Features

This section describes the features of the Application Protector Java.

Supported Java Distributions

The AP Java supports the following Java distributions:

- Java by Oracle Corporation, versions 1.8 and later
- Open JRE, versions 1.8 and later
- IBM J9, versions 1.8 and later

Trusted Applications

The AP Java can be accessed only by the trusted applications. Any application that protects, unprotects, or reprotects data must first be created as a trusted application.

For more information about how to make an application trusted, refer to the section *Creating a Trusted Application* in the *Policy Management Guide 9.1.0.4*.

Session Validity

A session is only valid until the session timeout that is passed as a parameter to the *createSession* API. The default validity of a session is *15* minutes. An active session is renewed every time the session is used.

Audit Logs

The AP Java Session API checks sessions to handle the generation of audit records and provide information on the error in the preceding or last session. Each session generates an audit record for every new protection method call combined with the data element used.

For single data item calls, a total of three audit log events are generated if you perform the following operations:

- One protect operation with data element name *a* (with count as 1)
- Five protect operations with data element *b* (with count as 5)
- One thousand unprotect operations with data element *a* (with count as 1000)

The same behavior is seen with Bulk data items, every data protection operation, such as, protect, unprotect, and reprotect will generate audit log events. For example, if the bulk size is 3 and a total of two bulk protect operations, having same data elements and session, are performed then one audit log will be generated with count as 6.

You can use this knowledge to ensure how you want the audit records to be generated, in case of single or bulk data item calls, by deciding how long a session is valid and how often new sessions are created using the *createSession* API.

An audit log for the trusted application is displayed each time an application is initialized, indicating whether initialization was successful or not. Note that audits are created only after *jpeplite.plm* file are loaded. The audits are generated in the ESA Forensics for easy access by the Security Officer.

Error Handling

If the AP Java is used to perform a security operation on bulk data, then an exception appears for all errors except for the error codes *22, 23, and 44*. Instead, an error list is returned for the individual items in the bulk data.

For more information about the AP Java API return codes, refer to the section *Application Protectors API Return Codes* in the *Protegrity Troubleshooting Guide 9.1.0.4*.

4.2 Architecture and Components

This section describes the architecture, the individual components, and the workflow of the Protegrity Application Protector (AP) Java solution.

The following figure shows the deployment architecture for the AP Java.

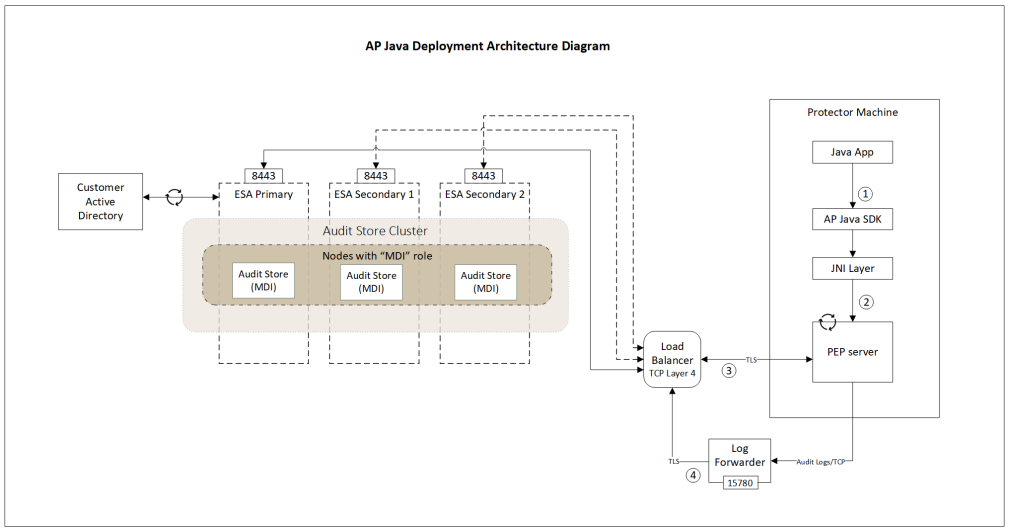

Figure 4-1: AP Java Deployment Architecture

The following table describes the components of the AP Java deployment architecture.

Table 4-1: Components of AP Java Deployment Architecture

Component	Description
Java App	Customer Java Application
AP Java SDK	Protegrity Application Protector Java SDK
Java Native Interface (JNI) layer	Interface between the Java SDK and the C layer
PEP Server	Component that downloads the policy package from the ESA and reads the package to perform any operations, such as, protect, unprotect, and reprotect
Enterprise Security Administrator (ESA)	Component of the Data Security Platform that consists of the following: • ESA Primary - Manages the policies, keys, monitoring, auditing and reporting of the protected systems in the enterprise • ESA Secondary 1 and 2 - Additional ESAs for Disaster Recovery
Log Forwarder	Log processing tool that collects the logs from the protectors and forwards them to the Audit Store Cluster
Load Balancer	Component that efficiently distributes the calls from the protector to the ESA
Customer Active Directory	Customer data that synchronizes with the ESA

The following steps describe the workflow of a sample AP Java deployment in the production environment.

1. The customer Java application initializes the Protegrity AP Java SDK.
2. During the initialization process, the PEP server synchronizes with the ESA to download the policy over a TLS channel and stores it in the memory.
3. The protector performs the data security operations, such as, protect, unprotect, or reprotect, using the downloaded policy.
4. The audit logs generated while performing the operations are forwarded to the ESA using the Log Forwarder.

4.3 Working with the PEP Server

The Policy Enforcement Point (PEP) server is a component that downloads the policy package from the ESA and reads the package to perform any operations, such as, protect, unprotect, and reprotect. It is a communication agent that accepts the policy deployed from the Hub Controller.

The Policy Enforcement Agent is responsible for the actual enforcement of the policy and protection of sensitive data. Its implementation differs based on the host technology what the protector protects - application, database, or file. The ESA allows security officers to easily specify data security requirements and distribute them across the enterprise to be executed locally. Once the policy is determined and set in the ESA, it is deployed to Protegrity Data Protectors for enforcement on their installed systems, ensuring consistent security enterprise-wide.

For more information about the various configurations of the PEP server using the *pepserver.cfg* file, refer to the section *Appendix: PEP Server Configuration File* in the *Installation Guide 9.1.0.4*.

4.4 Working with the Log Forwarder

The Log Forwarder component is a log processing tool that collects the data security operation logs from the Application Protector and forwards them to the Audit Store on the ESA.

The Log Forwarder uses ports, such as *15780*, which is configurable to transport protection and audit logs to the ESA. The ESA receives the logs and stores it in the Audit Store.

For more information about the Log Forwarder, refer to the section *Components of the Logging Architecture* in the *Protegrity Log Forwarding Guide 9.1.0.1*.

4.5 Link to JavaDoc

This section describes how you can access the JavaDoc that is included with the AP Java.

All APIs have been documented and explained in the JavaDoc that is included with the AP Java.

Table 4-2: Location of JavaDoc

OS	Location
Linux/Unix	`/opt/protegrity/applicationprotector/java/doc`
Windows	`C:\Program Files\Protegrity\Defiance AP\java\doc`

4.6 Running AP Java - Example

This section describes how to use the AP Java protection, unprotection, and reprotection APIs. It is assumed that the ESA is already available, and the AP Java has been successfully installed.

The tasks can be divided in the following order.

1. Create the data elements and data store in the Policy Management on the ESA Web UI.
2. Create the member sources and roles.
3. Configure the policy.
4. Configure the trusted application.
5. Add a trusted application to the data store.
6. Install the AP Java.

7. Use the protection and unprotection APIs by first protecting the string array and then unprotecting the protected value.

For more information on running the AP Java sample application, refer to the section *Running IAP - Example*.

Chapter 5

Application Protector Python

The Protegrity Application Protector (AP) Python provides APIs that integrate with the customer application to protect, unprotect, and reprotect sensitive data. The AP Python can be used with any customer application that is developed using the Python programming language.

> **Note:**
> The AP Python Protector has been updated in the build 9.1.0.0.x and this version of the document applies to the AP Python Protector 9.1.0.0.x and higher.
>
> If you are using an earlier version of the AP Python Protector, then refer to the document included in the documentation package with the build prior to 9.1.0.0.x.

The AP Python has the following protection and security access methods:

- Get product version
- Check access rights for users
- Get current key id
- Extract key id from data
- Get default data element
- Protect
- Unprotect
- Reprotect

For more information about the AP Python protection and security access methods, refer to the section *Application Protector (AP) Python APIs* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.0*.

5.1 Features

The following are the main features of AP Python:

- **Supported Python Distributions**

  The AP Python supports the following Python versions:

  - Python 3.7, 3.8, 3.9, 3.10, and 3.11 for Linux operating system
  - Python 3.7 for Windows operating system
- **Trusted Applications**

  AP Python APIs can be accessed only by trusted applications. Any application that protects or unprotects data must first be created as a Trusted Application.

  For more information about making an application trusted, refer to the section *Working with Trusted Applications* in the *Policy Management Guide 9.1.0.4*.

- **Session Validity**

  A session is only valid until the session timeout that is passed as a parameter to the *create_session* API. The default validity of a session is 15 minutes. An active session is renewed every time the session is used.

- **Session Handling**

  Sessions are needed to handle audit record generation. A session is valid for a specific time, and it is managed by the *timeout* value passed during the *create_session()* method. By default, the session timeout value is set to 15 minutes. For every call to the *create_session()* method, a new session object is created - a pool of session objects is not maintained. Python's garbage collector is used for destroying the Session objects once they are out of scope. You can also use the session object as Python's Context manager using the with statement.

  A session is automatically renewed every time it is used. Thus, for each call to a data protection operation, such as, protect, unprotect, and reprotect, the time for the session to remain alive is renewed.

  Audit Logs

  The AP Python API checks sessions to handle the generation of audit records. Each session generates an audit record for every new protection method call combined with the data element used.

  For single data item calls, a total of three audit log events are generated if you perform the following operations:
  - One protect operation with data element name *a* (with count as 1)
  - Five protect operations with data element *b* (with count as 5)
  - One thousand unprotect operations with data element *a* (with count as 1000)

  The same behavior is seen with Bulk data items, every data protection operation, such as, protect, unprotect, and reprotect will generate audit log events. For example, if the bulk size is 3 and a total of two bulk protect operations, having same data elements and session, are performed then one audit log will be generated with count as 6.

  You can use this knowledge to ensure how you want the audit records to be generated, in case of single / bulk data item calls , by deciding how long a session is valid and how often new sessions are created using create_session().

  Logs for the Trusted Application are an addition to the list of audit logs. A log for the same is available when you select the **Successful Audit** checkbox in the **Trusted Applications** window of the **Policy Management** > **Policies & Trusted Applications** menu in the ESA Web UI. The audits are created only after the AP Python APIs are invoked. Audits are generated in the ESA Forensics for easy access by the Security Officer.

For more information about Trusted Applications, refer to the section *Working with Trusted Applications* in the *Policy Management Guide 9.1.0.4*.

> **Note:**
> The Client IPs for all protection logs displays a default value of *0.0.0.0*.

> **Caution:**
> When a short running application has completed its execution (in less than a second), the audit logs will not be seen. In such cases, the session.flush_audits() API needs to be invoked.
>
> For more information about flush_audits, refer to the section *flush_audits* of the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

- **Error Handling**

  The AP Python APIs return an error message in case of policy and system error. If AP Python is used to perform a security operation on a single data item, then an exception appears in case of any error. Similarly, if AP Python is used to perform a security operation on bulk data, then an exception appears for all errors except the error codes *22, 23, and 44*. Instead, an error list is returned for the individual items in the bulk data for error codes 22, 23, and 44.

  For more information about the AP Python API error return codes, refer to the section *Application Protector (AP) Python API Return Codes* in the *Protegrity Troubleshooting Guide 9.1.0.4*.

- **Support for running AP Python in a Development Environment**

  The AP Python provides support for running it in a development environment. In this mode, customers can use the AP Python APIs along with a set of sample users and data elements that can be used to simulate the behavior of the APIs in production environment. This mode is also known as AP Python mock implementation. Customers can use this mode to test the integration of their applications with AP Python.

> **Note:** For more information on how to run AP Python in a development environment, refer to the section *Using AP Python in a Development Environment* in the *APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

5.2 Architecture and Components

This section describes the architecture, the individual components, and the workflow of the Protegrity Application Protector (AP) Python solution.

The following figure shows the deployment architecture for the AP Python.

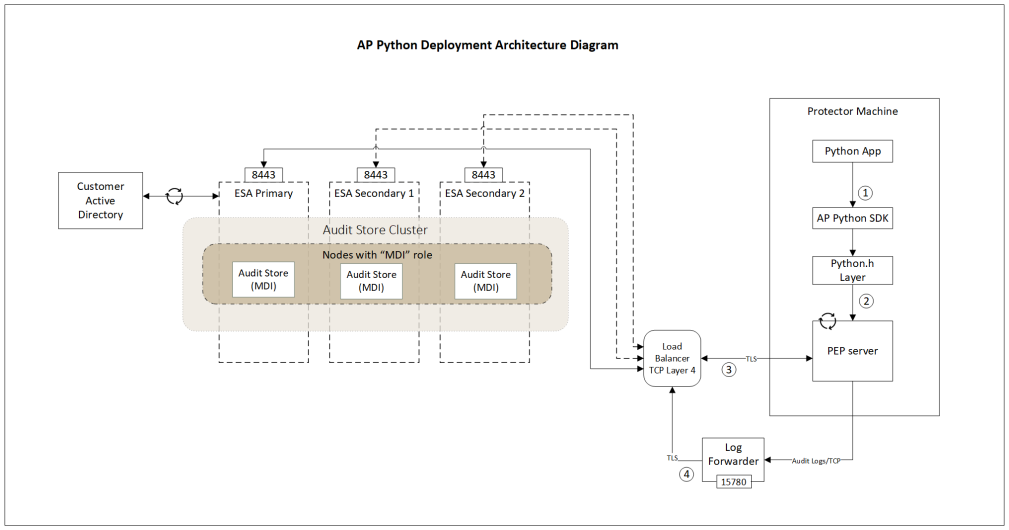

Figure 5-1: AP Python Deployment Architecture

The following table describes the components of the AP Python deployment architecture.

Table 5-1: Components of AP Python Deployment Architecture

Component	Description
Python App	Customer Python Application
AP Python SDK	Protegrity Application Protector Python SDK
Python.h Native layer	Interface between the Python SDK and the C layer
PEP Server	Component that downloads the policy package from the ESA and reads the package to perform any operations, such as, protect, unprotect, and reprotect
Enterprise Security Administrator (ESA)	Component of the Data Security Platform that consists of the following: • ESA Primary - Manages the policies, keys, monitoring, auditing and reporting of the protected systems in the enterprise • ESA Secondary 1 and 2 - Additional ESAs for Disaster Recovery
Log Forwarder	Log processing tool that collects the logs from the protectors and forwards them to the Audit Store Cluster
Load Balancer	Component that efficiently distributes the calls from the protector to the ESA
Customer Active Directory	Customer data that synchronizes with the ESA

The following steps describe the workflow of a sample AP Python deployment in the production environment.

1. The customer Python application initializes the Protegrity AP Python SDK.
2. During the initialization process, the PEP server synchronizes with the ESA to download the policy over a TLS channel and stores it in the memory.
3. The protector performs the data security operations, such as, protect, unprotect, or reprotect, using the downloaded policy.
4. The audit logs generated while performing the operations are forwarded to the ESA using the Log Forwarder.

5.3 Working with the PEP Server

The Policy Enforcement Point (PEP) server is a component that downloads the policy package from the ESA and reads the package to perform any operations, such as, protect, unprotect, and reprotect. It is a communication agent that accepts the policy deployed from the Hub Controller.

The Policy Enforcement Agent is responsible for the actual enforcement of the policy and protection of sensitive data. Its implementation differs based on the host technology what the protector protects - application, database, or file. The ESA allows

security officers to easily specify data security requirements and distribute them across the enterprise to be executed locally. Once the policy is determined and set in the ESA, it is deployed to Protegrity Data Protectors for enforcement on their installed systems, ensuring consistent security enterprise-wide.

For more information about the various configurations of the PEP server using the *pepserver.cfg* file, refer to the section *Appendix: PEP Server Configuration File* in the *Installation Guide 9.1.0.4*.

5.4 Working with the Log Forwarder

The Log Forwarder component is a log processing tool that collects the data security operation logs from the Application Protector and forwards them to the Audit Store on the ESA.

The Log Forwarder uses ports, such as *15780*, which is configurable to transport protection and audit logs to the ESA. The ESA receives the logs and stores it in the Audit Store.

For more information about the Log Forwarder, refer to the section *Components of the Logging Architecture* in the *Protegrity Log Forwarding Guide 9.1.0.1*.

5.5 Link to PyDoc

All APIs have been documented and explained in the PyDoc that is included with the AP Python.

To access the PyDoc on the Linux or Unix platform:

1. Run the following command to extract the AP Python setup file:

   ```
   tar -xvf APPython<Python version supported>Setup_<OS>_x64_<version>.tar
   ```

   The *appython-<version>* directory is extracted.

2. Navigate to the `appython-<version>/docs/html` directory.
3. Open the *index.html* file in a browser to access the AP Python Pydoc.

> **Note:**
> If you are setting up the AP Python in a virtual Linux or Unix environment, then convert the *appython-<version>/docs* directory to a zip file, download it locally, and then open the *index.html* in a browser to access the AP Python Pydoc.

To access the PyDoc on the Windows platform:

1. Extract the following AP Python setup file.
   APPythonDevSetup_Windows_x64_<version>.tar

   The *APPythonDevSetup_Windows_x64_<version>* directory is extracted.

2. Navigate to the `APPythonSetup_Windows_x64_<version>\dist\appython-<version>.tar` directory.
3. Extract the *appython-<version>.tar* in the `\dist` directory.
   The *appython-<version>* directory is extracted.

4. Navigate to the `appython-<version>\docs\index` directory.

5. Open the *index.html* file in a browser to access the AP Python Pydoc.

5.6 AP Python APIs and Supported Protection Methods

This section describes the AP Python APIs and the protection methods supported by it.

The following table lists the AP Python APIs, the input and output data types, and the supported Protection Methods.

Table 5-2: AP Python APIs and Supported Protection Methods

Operation	Input	Output	Protection Method Supported
Protect	String	String	Tokenization, No Encryption, FPE
Protect	String	Bytes*1	Encryption, CUSP
Protect	Integer*3 *4	Integer*3 *4	Tokenization, No Encryption
Protect	Integer	Bytes*1	Encryption, CUSP
Protect	Float	Float	No Encryption
Protect	Float	Bytes*1	Encryption, CUSP
Protect	Date	Date	Tokenization, No Encryption
Protect	Bytes	Bytes*1	Tokenization, Encryption, No Encyption, CUSP, Printable, Binary, Hashing
Unprotect	String	String	Tokenization, No Encryption, FPE
Unprotect	Bytes*2	String	Encryption, CUSP
Unprotect	Integer*3 *4	Integer*3 *4	Tokenization, No Encryption
Unprotect	Bytes*2	Integer	Encryption, CUSP
Unprotect	Float	Float	No Encryption
Unprotect	Bytes*2	Float	Encryption, CUSP
Unprotect	Date	Date	Tokenization, No Encryption
Unprotect	Bytes*2	Bytes	Tokenization, Encryption, No Encyption, CUSP, Printable, Binary, Hashing
Reprotect	String	String	Tokenization, No Encryption, FPE
Reprotect	Integer*3 *4	Integer*3 *4	Tokenization
Reprotect	Float	Float	No Encryption
Reprotect	Bytes	Bytes*1	Tokenization, Encryption, CUSP, Printable, Binary, Hashing **Note:** If a protected value is generated using *Byte* as both *Input* and *Output*, then only Encryption/CUSP is supported.

Note:

*1 - You must pass the `encrypt_to=bytes` keyword argument to the API for encrypting the data.

*2 - You must pass the *decrypt_to* parameter to the API for decrypting the data and set its value to the data type of the original data.

[*3] – The integer token type with 2 bytes as input is supported for the AP Python. An additional parameter - *int_size=2*, needs to be passed to perform the protect, unprotect, or reportect operation. For a bulk call to the protect, unprotect, or reprotect APIs, the protected, unprotected, or reprotected data and the success return code, which is (*6* for successful protect operation), (*8* for successful unprotect operation), and (*50* for successful reprotect operation) are returned. For a single call to the protect, unprotect, or reprotect APIs, the protected, uprotected, or reprotected data is returned.

[*4] - If the user passes 4-byte integer (values ranging from -2,147,483,648 to +2,147,483,647) as data and uses the 8-byte Integer token type data element as input for the protect, unprotect, or reprotect APIs, then the data protection operation will not be successful. For a Bulk call using the protect, unprotect, and reprotect APIs, the error code, ***44***, appears. For a single call using the protect, unprotect, and reprotect APIs, an exception will be thrown and the error message, `44, Content of input data is not valid` appears.

5.7 Running AP Python - Example

This section describes how to use the AP Python protection, unprotection, and reprotection APIs. It is assumed that the ESA is already available, and the AP Python has been successfully installed.

The tasks can be divided in the following order.

1. Create the data elements and data store in the Policy Management on the ESA Web UI.
2. Create the member sources and roles.
3. Configure the policy.
4. Configure the trusted application.
5. Add a trusted application to the data store.
6. Install the AP Python.
7. Use the protection and unprotection APIs by first protecting the string array and then unprotecting the protected value.

For more information on running the AP Python sample application, refer to the section *Running IAP - Example*.

Chapter 6

Application Protector Golang

The Protegrity Application Protector (AP) Golang (Go) provides APIs that can be utilized with Golang to protect, unprotect, or reprotect sensitive data. The AP Go can be integrated with any customer application that is developed using the Golang programming language.

> **Note:**
> The AP Go Protector has been updated in the build 9.1.0.0.x and this version of the document applies to the AP Go Protector 9.1.0.0.x and higher.
>
> If you are using an earlier version of the AP Go Protector, then refer to the document included in the documentation package with the build prior to 9.1.0.0.x.

The AP Go has the following protection and security access methods:

- Get product version
- Check access rights for users
- Get current key id for data element
- Get key id from protected data
- Get default data element
- Protect
- Unprotect
- Reprotect

For more information about the AP Go protection and security access methods, refer to the section *Application Protector (AP) Golang APIs* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.0*.

6.1 Features

This section describes the features of the Application Protector Golang.

- **Supported Golang Distributions**

The AP Go supports the Golang 1.x for all operating systems.

- **Trusted Applications**

  The AP Go can be accessed only by the trusted applications. Any application that protects or unprotects data must first be created as a Trusted Application.

  For more information about how to make an application trusted, refer to the section *Creating a Trusted Application* in the *Policy Management Guide 9.1.0.4*.

- **Session Validity**

  A session is only valid until the session timeout that is passed as a parameter to the *NewSession()* API. The default validity of a session is *15* minutes. An active session is renewed every time the session is used.

- **Session Handling**

  Sessions are needed to handle the audit record generation. A session is valid for a specific time, and it is managed by the *timeout* value passed during the *NewSession()* method. By default, the session timeout value is set to *15* minutes. For every call to the *NewSession()* method, a new session object is created - a pool of session objects is not maintained. Golang's garbage collector is used for destroying the Session objects once they are out of scope.

  A session is automatically renewed every time it is used. Thus, for each call to a data protection operation, such as, protect, unprotect, and reprotect, the time for the session to remain active is renewed.

- **Audit Logs**

  The AP Go Session API checks sessions to handle the generation of audit records and provide information on the error in the preceding or last session. Each session generates an audit record for every new protection method call combined with the data element used.

  For single data item calls, a total of three audit log events are generated if you perform the following operations:

  - One protect operation with data element name *a* (with count as 1)
  - Five protect operations with data element *b* (with count as 5)
  - One thousand unprotect operations with data element *a* (with count as 1000)

  The same behavior is seen with Bulk data items, every data protection operation, such as, protect, unprotect, and reprotect will generate audit log events. For example, if the bulk size is 3 and a total of two bulk protect operations, having same data elements and session, are performed then one audit log will be generated with count as 6.

  You can use this knowledge to ensure how you want the audit records to be generated, in case of single or bulk data item calls, by deciding how long a session is valid and how often new sessions are created using the *NewSession()* API.

  > **Caution:**
  > When a short running application has completed its execution (in less than a second), the audit logs will not be seen. In such cases, the *session.FlushAudits()* API needs to be invoked.
  >
  > For more information about the FlushAudits API, refer to the section *FlushAudits() API* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

An audit log for the trusted application is displayed each time an application is initialized, indicating whether initialization was successful or not. The audits are created only after the AP Go APIs are invoked. The audits are generated in the ESA Forensics for easy access by the Security Officer.

6.2 Architecture and Components

This section describes the architecture, the individual components, and the workflow of the Protegrity Application Protector (AP) Golang (Go) solution.

The following figure shows the deployment architecture for the AP Go.

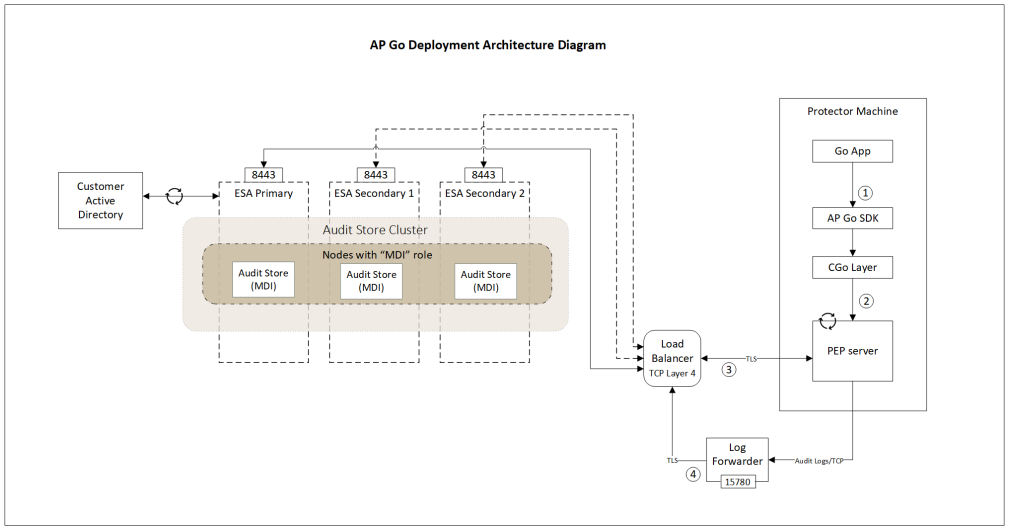

Figure 6-1: AP Go Deployment Architecture

The following table describes the components of the AP Go deployment architecture.

Table 6-1: Components of AP Go Deployment Architecture

Component	Description
Golang App	Customer Golang Application
AP Golang SDK	Protegrity Application Protector Golang SDK
CGo layer	Interface between the Golang SDK and the C layer
PEP Server	Component that downloads the policy package from the ESA and reads the package to perform any operations, such as, protect, unprotect, and reprotect
Enterprise Security Administrator (ESA)	Component of the Data Security Platform that consists of the following: • ESA Primary - Manages the policies, keys, monitoring, auditing and reporting of the protected systems in the enterprise • ESA Secondary 1 and 2 - Additional ESAs for Disaster Recovery
Log Forwarder	Log processing tool that collects the logs from the protectors and forwards them to the Audit Store Cluster
Load Balancer	Component that efficiently distributes the calls from the protector to the ESA
Customer Active Directory	Customer data that synchronizes with the ESA

The following steps describe the workflow of a sample AP Go deployment in the production environment.

1. The customer Go application initializes the Protegrity AP Go SDK.

2. During the initialization process, the PEP server synchronizes with the ESA to download the policy over a TLS channel and stores it in the shared memory.
3. The protector performs the data security operations, such as, protect, unprotect, or reprotect, using the downloaded policy.
4. The audit logs generated while performing the operations are forwarded to the ESA using the Log Forwarder.

6.3 Working with the PEP Server

The Policy Enforcement Point (PEP) server is a component that downloads the policy package from the ESA and reads the package to perform any operations, such as, protect, unprotect, and reprotect. It is a communication agent that accepts the policy deployed from the Hub Controller.

The Policy Enforcement Agent is responsible for the actual enforcement of the policy and protection of sensitive data. Its implementation differs based on the host technology what the protector protects - application, database, or file. The ESA allows security officers to easily specify data security requirements and distribute them across the enterprise to be executed locally. Once the policy is determined and set in the ESA, it is deployed to Protegrity Data Protectors for enforcement on their installed systems, ensuring consistent security enterprise-wide.

For more information about the various configurations of the PEP server using the *pepserver.cfg* file, refer to the section *Appendix: PEP Server Configuration File* in the *Installation Guide 9.1.0.4*.

6.4 Working with the Log Forwarder

The Log Forwarder component is a log processing tool that collects the data security operation logs from the Application Protector and forwards them to the Audit Store on the ESA.

The Log Forwarder uses ports, such as *15780*, which is configurable to transport protection and audit logs to the ESA. The ESA receives the logs and stores it in the Audit Store.

For more information about the Log Forwarder, refer to the section *Components of the Logging Architecture* in the *Protegrity Log Forwarding Guide 9.1.0.1*.

6.5 Running AP Go - Example

This section describes how to use the AP Go protection, unprotection, and reprotection APIs. It is assumed that the ESA is already available, and the AP Go has been successfully installed.

The tasks can be divided in the following order.

1. Create the data elements and data store in the Policy Management on the ESA Web UI.
2. Create the member sources and roles.
3. Configure the policy.
4. Configure the trusted application.
5. Add a trusted application to the data store.
6. Install the AP Go.
7. Use the protection and unprotection APIs by first protecting the string array and then unprotecting the protected value.

For more information on running the AP Go sample application, refer to the section *Running IAP - Example*.

Chapter 7

Application Protector NodeJS

The Protegrity Application Protector (AP) NodeJS provides APIs that integrate with the customer application to protect, unprotect, and reprotect sensitive data. The AP NodeJS can be used with any customer application that is developed for NodeJS.

> **Note:**
> The AP NodeJS Protector has been updated in the build 9.1.0.0.x and this version of the document applies to the AP NodeJS Protector 9.1.0.0.x and higher.

The AP NodeJS has the following protection and security access methods:

- Get product version
- Check Access
- Protect
- Unprotect
- Reprotect

For more information about the AP NodeJS protection and security access methods, refer to the section *Application Protector (AP) NodeJS APIs* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.0*.

7.1 Features

This section describes the features of the Application Protector NodeJS.

Supported NodeJS Distributions

The AP NodeJS supports the following NodeJS distributions:

- NodeJS, version 12 or higher

Trusted Applications

The AP NodeJS can be accessed only by the trusted applications. Any application that protects, unprotects, or reprotects data must first be created as a trusted application.

For more information about how to make an application trusted, refer to the section *Creating a Trusted Application* in the *Policy Management Guide 9.1.0.4*.

Audit Logs

The AP NodeJS generates an audit record for every new protection method call combined with the data element and user name used.

For single data item calls, a total of three audit log events are generated if you perform the following operations:

- One protect operation with data element name *a* (with count as 1)
- Five protect operations with data element *b* (with count as 5)
- One thousand unprotect operations with data element *a* (with count as 1000)

The same behavior is seen with Bulk data items, every data protection operation, such as, protect, unprotect, and reprotect will generate audit log events. For example, if the bulk size is 3 and a total of two bulk protect operations, having same data elements and session, are performed then one audit log will be generated with count as 6.

Caution:

When a short running application has completed its execution (in less than a second), the audit logs will not be seen. In such cases, the *flushAudits()* API needs to be invoked.

For more information about the flushAudits API, refer to the section *flushAudits API* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

An audit log for the trusted application is displayed each time an application is initialized, indicating whether initialization was successful or not. The audits are generated in the ESA Forensics for easy access by the Security Officer.

7.2 Architecture and Components

This section describes the architecture, the individual components, and the workflow of the Protegrity Application Protector (AP) NodeJS solution.

The following figure shows the deployment architecture for the AP NodeJS.

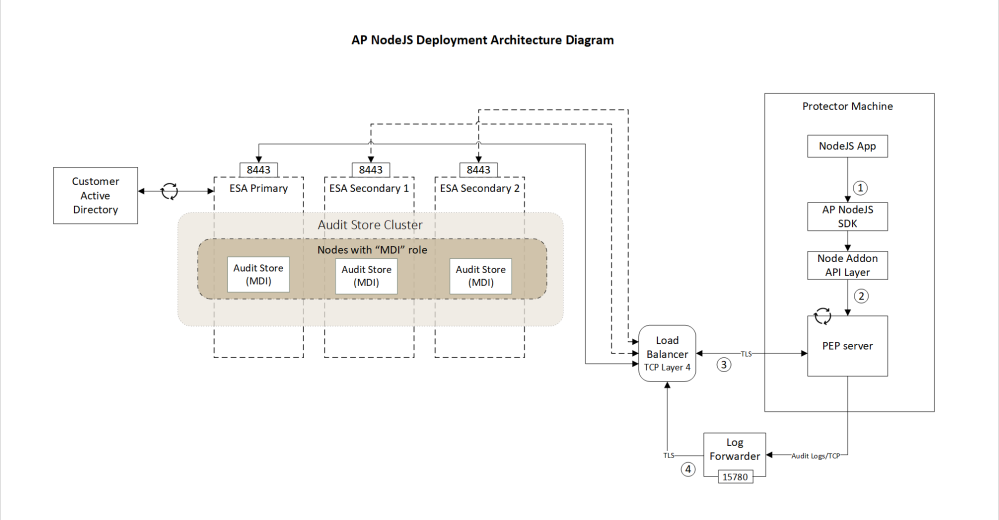

Figure 7-1: AP NodeJS Deployment Architecture

The following table describes the components of the AP NodeJS deployment architecture.

Table 7-1: Components of AP NodeJS Deployment Architecture

Component	Description
NodeJS App	Customer NodeJS Application.
AP NodeJS SDK	Protegrity Application Protector NodeJS SDK.
NodeJS Addon API Layer	Interface between the NodeJS SDK and the C layer.
PEP Server	Component that downloads the policy package from the ESA and reads the package to perform any operations, such as, protect, unprotect, and reprotect.
Enterprise Security Administrator (ESA)	Component of the Data Security Platform that consists of the following: • ESA Primary - Manages the policies, keys, monitoring, auditing and reporting of the protected systems in the enterprise. • ESA Secondary 1 and 2 - Additional ESAs for Disaster Recovery.
Log Forwarder	Log processing tool that collects the logs from the protectors and forwards them to the Audit Store Cluster.
Load Balancer	Component that efficiently distributes the calls from the protector to the ESA.
Customer Active Directory	Customer data that synchronizes with the ESA.

The following steps describe the workflow of a sample AP NodeJS deployment in the production environment.

1. The customer NodeJS application initializes the Protegrity AP NodeJS SDK.
2. During the initialization process, the PEP server synchronizes with the ESA to download the policy over a TLS channel and stores it in the memory.
3. The protector performs the data security operations, such as, protect, unprotect, or reprotect, using the downloaded policy.
4. The audit logs generated while performing the operations are forwarded to the ESA using the Log Forwarder.

7.3 Working with the PEP Server

The Policy Enforcement Point (PEP) server is a component that downloads the policy package from the ESA and reads the package to perform any operations, such as, protect, unprotect, and reprotect. It is a communication agent that accepts the policy deployed from the Hub Controller.

The Policy Enforcement Agent is responsible for the actual enforcement of the policy and protection of sensitive data. Its implementation differs based on the host technology what the protector protects - application, database, or file. The ESA allows

security officers to easily specify data security requirements and distribute them across the enterprise to be executed locally. Once the policy is determined and set in the ESA, it is deployed to Protegrity Data Protectors for enforcement on their installed systems, ensuring consistent security enterprise-wide.

For more information about the various configurations of the PEP server using the *pepserver.cfg* file, refer to the section *Appendix: PEP Server Configuration File* in the *Installation Guide 9.1.0.4*.

7.4 Working with the Log Forwarder

The Log Forwarder component is a log processing tool that collects the data security operation logs from the Application Protector and forwards them to the Audit Store on the ESA.

The Log Forwarder uses ports, such as *15780*, which is configurable to transport protection and audit logs to the ESA. The ESA receives the logs and stores it in the Audit Store.

For more information about the Log Forwarder, refer to the section *Components of the Logging Architecture* in the *Protegrity Log Forwarding Guide 9.1.0.1*.

7.5 Running AP NodeJS - Example

This section describes how to use the AP NodeJS protection, unprotection, and reprotection APIs. It is assumed that the ESA is already available, and the AP NodeJS has been successfully installed.

The tasks can be divided in the following order.

1. Create the data elements and data store in the Policy Management on the ESA Web UI.
2. Create the member sources and roles.
3. Configure the policy.
4. Configure the trusted application.
5. Add a trusted application to the data store.
6. Install the AP NodeJS.
7. Use the protection and unprotection APIs by first protecting the string array and then unprotecting the protected value.

For more information on running the AP NodeJS sample application, refer to the section *Running AP - Example*.

Chapter 8

Application Protector .Net

The Protegrity Application Protector (AP) .Net provides APIs that integrate with the customer application to protect, unprotect, and reprotect sensitive data. The AP .Net can be used with any customer application that is developed using the .Net Core and C# programming language.

> **Note:**
> The AP .Net Protector has been updated in the build 9.1.0.0.x and this version of the document applies to the AP .Net Protector 9.1.0.0.x and higher.

The AP .Net has the following protection and security access methods:

- Get product version
- Check Access
- Protect
- Unprotect
- Reprotect

For more information about the AP .Net protection and security access methods, refer to the section *Application Protector (AP) .Net APIs* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.0*.

8.1 Features

This section describes the features of the Application Protector .Net.

Supported .Net Distributions

The AP .Net supports the following:

- Supported .NET framework versions: 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
- Supported .NET Core versions: 6.0 onwards

Trusted Applications

The AP .Net can be accessed only by the trusted application. Any application that protects, unprotects, or reprotects data must first be configured as a trusted application.

For more information about how to make an application trusted, refer to the section *Creating a Trusted Application* in the *Policy Management Guide 9.1.0.4*.

Audit Logs

The AP .Net generates an audit record for every new protection method call combined with the data element and user name used.

For single data item calls, a total of three audit log events are generated if you perform the following operations:

- One protect operation with data element name *a* (with count as 1)
- Five protect operations with data element *b* (with count as 5)
- One thousand unprotect operations with data element *a* (with count as 1000)

The same behavior is seen with Bulk data items, every data protection operation, such as, protect, unprotect, and reprotect will generate audit log events. For example, if the bulk size is three and a total of two bulk protect operations, having same data elements and session, are performed then one audit log will be generated with count as six.

Caution:

If a short running application has completed its execution (in less than a second), then the audit logs will not be seen. In such cases, the *FlushAudits()* API needs to be invoked.

It is recommended to invoke *FlushAudits* API at the point where the application exits.

If the *FlushAudits* API is invoked after every operation, then the performance of the protector might get impacted.

For more information about the FlushAudits API, refer to the section *FlushAudits API* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

An audit log for the trusted application is displayed each time an application is initialized, indicating whether initialization was successful or not. The audits are generated in the ESA Forensics for easy access by the Security Officer.

8.2 Architecture and Components

This section describes the architecture, the individual components, and the workflow of the Protegrity Application Protector (AP) .Net solution.

The following figure shows the deployment architecture for the AP .Net.

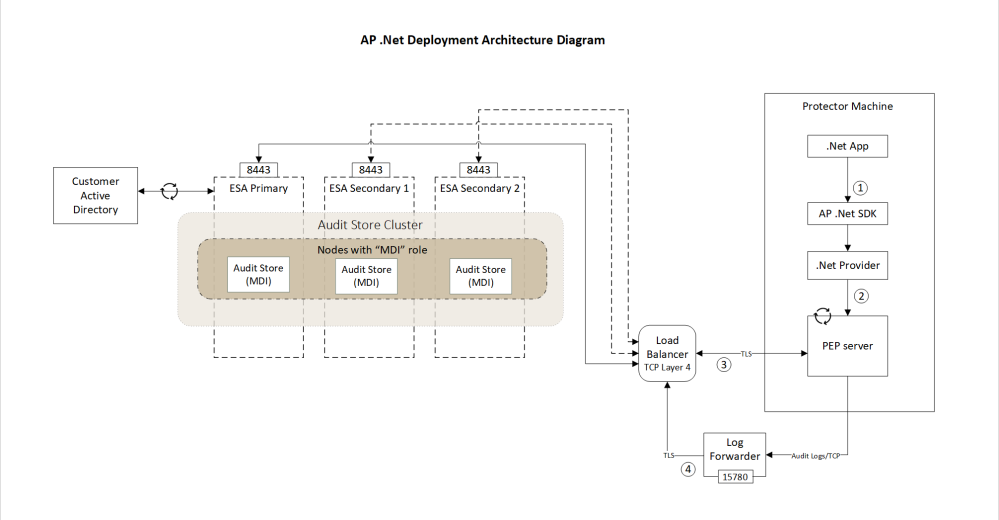

Figure 8-1: AP .Net Deployment Architecture

The following table describes the components of the AP .Net deployment architecture.

Table 8-1: Components of AP .Net Deployment Architecture

Component	Description
.Net App	Customer .Net Application.
AP .Net SDK	Protegrity Application Protector .Net SDK.
.Net Provider	Interface between the .Net SDK and the C layer.
PEP server	Component that downloads the policy package from the ESA and reads the package to perform any operations, such as, protect, unprotect, and reprotect.
Enterprise Security Administrator (ESA)	Component of the Data Security Platform that consists of the following: • ESA Primary - Manages the policies, keys, monitoring, auditing and reporting of the protected systems in the enterprise. • ESA Secondary 1 and 2 - Additional ESAs for Disaster Recovery.
Log Forwarder	Log processing tool that collects the logs from the protectors and forwards them to the Audit Store Cluster.
Load Balancer	Component that efficiently distributes the calls from the protector to the ESA.
Customer Active Directory	Customer data that synchronizes with the ESA.

The following steps describe the workflow of a sample AP .Net deployment in the production environment.

1. The customer .Net application initializes the Protegrity AP .Net SDK.
2. During the initialization process, the PEP server synchronizes with the ESA to download the policy over a TLS channel and stores it in the memory.
3. The protector performs the data security operations, such as, protect, unprotect, or reprotect, using the policy present in the memory.
4. The audit logs generated while performing the operations are forwarded to the ESA using the Log Forwarder.

8.3 Working with the PEP Server

The Policy Enforcement Point (PEP) server is a component that downloads the policy package from the ESA and reads the package to perform any operations, such as, protect, unprotect, and reprotect. It is a communication agent that accepts the policy deployed from the Hub Controller.

The Policy Enforcement Agent is responsible for the actual enforcement of the policy and protection of sensitive data. Its implementation differs based on the host technology what the protector protects - application, database, or file. The ESA allows security officers to easily specify data security requirements and distribute them across the enterprise to be executed locally. Once the policy is determined and set in the ESA, it is deployed to Protegrity Data Protectors for enforcement on their installed systems, ensuring consistent security enterprise-wide.

For more information about the various configurations of the PEP server using the *pepserver.cfg* file, refer to the section *Appendix: PEP Server Configuration File* in the *Installation Guide 9.1.0.4*.

8.4 Working with the Log Forwarder

The Log Forwarder component is a log processing tool that collects the data security operation logs from the Application Protector and forwards them to the Audit Store on the ESA.

The Log Forwarder uses ports, such as *15780*, which is configurable to transport protection and audit logs to the ESA. The ESA receives the logs and stores it in the Audit Store.

For more information about the Log Forwarder, refer to the section *Components of the Logging Architecture* in the *Protegrity Log Forwarding Guide 9.1.0.1*.

8.5 Running AP .Net - Example

This section describes how to use the AP .Net protection, unprotection, and reprotection APIs. It is assumed that the ESA is already available, and the AP .Net has been successfully installed.

The tasks can be divided in the following order.

1. Create the data elements and data store in the Policy Management on the ESA Web UI.
2. Create the member sources and roles.
3. Configure the policy.
4. Configure the trusted application.
5. Add a trusted application to the data store.
6. Install the AP .Net.
7. Use the protection and unprotection APIs by first protecting the string array and then unprotecting the protected value.

For more information on running the AP .Net sample application, refer to the section *Running AP - Example*.

Chapter 9

Running AP - Example

This section provides an example on how to use the Application Protector protection, unprotection, and reprotection APIs. It is assumed that the ESA is already available, and the AP has been successfully installed.

The tasks can be divided in the following order.

1. Create the data elements and data store in the Policy Management on the ESA Web UI.
2. Create the member sources and roles.
3. Configure the policy.
4. Configure the trusted application.
5. Add a trusted application to the data store.
6. Install the Application Protector.
7. Use the protection and unprotection APIs by first protecting the string array and then unprotecting the protected value.

9.1 Creating Data Elements and Data Stores in Policy Management

This section describes how to create a data element and a data store in the ESA.

Before you begin
Before you run the application, decide on how you would like to protect the data – using encryption or tokenization. Protection and unprotection methods are available for both.

➤ To create a data element and a data store:

1. To create a data element, from the ESA Web UI, navigate to **Policy Management** > **Data Elements & Masks** > **Data Elements**.

   For more details about creating data elements, refer to the section *Working With Data Elements* in the *Policy Management Guide 9.1.0.4*.

2. Similarly, to create a data store, go to **Policy Management** > **Data Stores.**

   For more details about creating data elements and data stores, refer to the section *Working With Data Elements* in the *Policy Management Guide 9.1.0.4*.

9.2 Creating Member Sources and Roles

This section describes how to create a member source and a role in the ESA.

➤ To create the member sources and roles:

1. To create a member source, from the ESA Web UI, navigate to **Policy Management** > **Roles & Member Sources** > **Member Sources**.

   For more information about creating the member sources, refer to the section *Working with Member Sources* in the *Policy Management Guide 9.1.0.4*.

2. To create a role, from the ESA Web UI, navigate to **Policy Management** > **Roles & Member Sources** > **Roles**.

   For more information about creating the roles, refer to the section *Working with Roles* in the *Policy Management Guide 9.1.0.4*.

9.3 Configuring a Policy

This section describes how to configure a policy in the ESA.

➤ To configure a new policy:

1. From the ESA Web UI, navigate to **Policy Management** > **Policies & Trusted Applications** > **Policies**.
2. Click **Add New Policy**.

   The **New Policy** screen appears.
3. After the policy is configured for the application user, add the permissions, data elements, roles, and data stores to the policy and then save it.
4. Deploy the policy using the Policy Management Web UI.

   For more information about creating a data security policy, refer to the section *Creating and Deploying Policies* in the *Policy Management Guide 9.1.0.4*.

9.4 Configuring a Trusted Application

This section describes how to configure a trusted application in the ESA.

Before you begin

To ensure that only the applications and users configured in the ESA are able to access the AP APIs, these have to be configured as trusted applications under the ESA security policy. If a policy is deployed but the application or the user is not trusted,

then while performing any protect or unprotect operations, an error message - `API consumer is not part of the trusted applications, please contact the Security Officer` - appears and the AP aborts.

> **Note:** The AP C does not support trusted application.

➤ To configure a new trusted application:

1. From the ESA Web UI, navigate to **Policy Management** > **Policies & Trusted Applications** > **Trusted Application**.
2. Create a trusted application.
3. Deploy the trusted application using the Policy Management Web UI.

   For more information about the trusted applications, refer to the section *Working with Trusted Applications* in the *Policy Management Guide 9.1.0.4*.

9.5 Adding a Trusted Application to a Data Store

This section describes how to add a trusted application to a data store in the ESA.

➤ To add a trusted application to a data store:

1. From the ESA Web UI, navigate to **Policy Management** > **Data Stores**.

   The list of all the data stores appear.
2. Select the required data store.

   The screen to edit the data store appears.
3. Under the **Trusted Applications** tab, click **Add**.

   The screen to add the trusted application appears.
4. Select the required trusted application and click **Add**.
5. Select the required policy and deploy it using the Policy Management Web UI.

   For more information about adding a trusted application to a data store, refer to the section *Adding Trusted Applications to the Data Store* in the *Policy Management Guide 9.1.0.4*.

9.6 Installing Application Protector

This section describes how to install Application Protector.

9.6.1 Installing AP C

This section describes how to install the Application Protector C.

➤ To install the AP C:

1. The AP C must be successfully installed to protect or unprotect sensitive information.

For more information about the installation process of the AP C, refer to the section *Installing Application Protector C* in the *Installation Guide 9.1.0.4*.

2. To verify that the AP C is installed correctly, run the *XCGetVersion* method to check the version of the installed AP C.

```
XCGetVersion( XC_CHAR* pszVersion, const XC_UINT4 ui4VersionLength );
```

9.6.2 Installing AP Java

This section describes how to install the Application Protector Java.

To install the AP Java:

1. The AP Java must be successfully installed to protect, unprotect, or reprotect sensitive information.

   For more information about the installation process of the AP Java, refer to the section *Installing Application Protector Java* in the *Installation Guide 9.1.0.4*.

2. To verify if the AP Java is successfully installed, perform the following steps.
   a. Initialize the AP Java.

      For more information about the AP Java initialization API, refer to the section *getProtector* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

   b. Run the *GetVersion* method using the following command to check the version of the installed AP Java.

      public java.lang.String getVersion()

      The following is a sample code to check the version number of the installed AP Java.

```
/* Illustrates how to call getVersion() api to know the version of Application protector
* Executing this for the first time creates a forensic entry that should be
* added to the authorized app
*
* Compiled as  : javac -cp ApplicationProtectorJava.jar AP_Java_getVersion
* Run as       : java -cp  ApplicationProtectorJava.jar AP_Java_getVersion
*
*/

import com.protegrity.ap.java.*;

public class AP_Java_getVersion {

    public static void main(String[] args) throws ProtectorException {

        Protector protector=null;
        try {
            protector=Protector.getProtector();
            System.out.println("Product version : "+protector.getVersion());
        } catch (ProtectorException e) {
            e.printStackTrace();
            throw e;
        }

   }
}
```

9.6.3 Installing AP Python

This section describes how to install the Application Protector Python.

To install the AP Python:

1. The AP Python must be successfully installed to protect or unprotect sensitive information.
   The AP Python installation package is installed using the *pip* installer available in Python.

   For more information about the installation process of the AP Python, refer to the section *Installing Application Protector (AP) Python* in the *Installation Guide 9.1.0.4*.

2. To verify if the AP Python is successfully installed, perform the following steps.
   a. Initialize the AP Python.
      For more information about the AP Python initialization API, refer to the section *Protector* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

   b. Run the *get_version* method using the following command to check the version of the installed AP Python.
      The following is a sample code to check the version number of the installed AP Python.

```
from appython import Protector
protector = Protector()
session = protector.create_session("User1")
print(protector.get_version())
```

   c. Save the sample code and name it *<filename>.py*.
   d. Run the command **`python <filename>.py`** to know the version of the installed AP Python.

9.6.4 Installing AP Go

This section describes how to install the Application Protector Go.

To install the AP Go:

1. The AP Go must be successfully installed to protect or unprotect sensitive information.
   For more information about the installation process of the AP Go, refer to the section *Installing Application Protector Golang* in the *Installation Guide 9.1.0.4*.

2. To verify that the AP Go is installed correctly, run the *GetVersion()* method using the following method to check the version of the installed AP Go.
   func GetVersion() string
   The following is a sample code to check the version number of the installed AP Go.

```
package main

import (
    "fmt"

    apgo "protegrity.com/apgo"
)

func main() {
    fmt.Printf("AP Go Version: %s\n\n", apgo.GetVersion())
}
```

9.6.5 Installing AP NodeJS

This section describes how to install the Application Protector NodeJS.

To install the AP NodeJS:

1. The AP NodeJS must be successfully installed to perform the protect, unprotect, or reprotect operations.

   For more information about the installation process of the AP NodeJS, refer to the section *Installing Application Protector NodeJS* in the *Installation Guide 9.1.0.4*.

2. To verify that the AP NodeJS is installed correctly, run the *getVersion* API to check the version of the installed AP NodeJS.

```
const protector = require('apnode')
const protectorVersion = protector.getVersion()
console.log("Protector Version :",protectorVersion)
```

9.6.6 Installing AP .Net

This section describes how to install the Application Protector .Net.

To install the AP .Net:

1. The AP .Net must be successfully installed to protect, unprotect, or reprotect sensitive information.

   For more information about the installation process of the AP .Net, refer to the section *Installing Application Protector .Net* in the *Installation Guide 9.1.0.4*.

2. To verify that the AP .Net is installed correctly, perform the following steps.
   a. Initialize the AP .Net.

      For more information about the AP .Net initialization API, refer to the section *GetProtector* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

   b. Run the *GetVersion* method using the following command to check the version of the installed AP .Net.

```
protector = Protector.GetProtector();
Console.WriteLine("Application Protector version: " + protector.GetVersion());
```

9.7 Using the Application Protector APIs

This section describes how to use the Application Protector using a sample application.

9.7.1 Using the AP C APIs

This section describes how to use AP C APIs using a sample application of the Linux and Windows platforms.

9.7.1.1 Using the AP C APIs on Linux

This section describes how to use the AP C APIs using a sample application on a Linux platform.

After you have setup the policy, you can begin testing the AP C APIs for protection, unprotection, and reprotection using the sample code.

To run the sample AP C code, perform the following steps.

1. Copy and rename *xcpep.plm* to *libxcpep.so* using the following command in the same directory as *<installation_directory>/defiance_xc/bin*.

   ```
   cp xcpep.plm libxcpep.so
   ```

2. Run the following commands:

   ```
   gcc -g -o sample -DUNIX -I /opt/protegrity/defiance_xc/include -L/opt/protegrity/defiance_xc/bin -lxcpep -lpthread sample.c

   export LD_LIBRARY_PATH=/opt/protegrity/defiance_xc/bin/

   ./sample -p parameter -u user -d1 dataelement1 [-d2 dataelement2] (-prot | -unprot | -reprot | -hmac | -check | -currkid dataelement | -getkid dataelement data) [-in=hex|raw|int] [-out=text|raw|int] [-op <operation>] [-cu certuser] [-cp certpassword] [-dir workingdir] [-batch records] [-version] [-defde policy] [-data data | NULL] [-scid seccoordID] [-f datafile]
   ```

Caution:

When a short running application has completed its execution (in less than a second), the audit logs will not be seen. In such cases, the *XCFlushPepAudits* function() API needs to be invoked.

For more information about the *XCFlushPepAudits* function, refer to the section *XCFlushPepAudits Function* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

For more information on the AP C methods, refer to section *Application Protector (AP) C APIs* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.0*.

The following is a sample application for the AP C.

```
/*
 * AP-C Sample code to protect some data.
 * Refer readme for compile and execute instructions.
 */

#include <stdio.h>
#include <string.h>

/* xcapi.h includes the AP-C API signature.
 * xcdefinitions.h includes limitations and structures.
 **/
#include "xcapi.h"

/**
 * Maximum growth factor for
 * Tokenization : 1.35 ( eg: Unicode Token )
 * FPE          : 2.00 ( eg: UTF-16 BE/LE  )
 * Encryption   : 512  ( eg: AES 128/256 and etc. )
 */

#define XC_MAX_INPUT_LEN  ((4*(4096+2)) + XC_BYTES_OVERHEAD )
#define XC_MAX_OUTPUT_LEN ((2*(XC_MAX_INPUT_LEN+1)) + XC_BYTES_OVERHEAD )


void printUsage()
{
  printf( "Usage: xcenc -p parameter -u user -d1 dataelement1 [-cu certuser] [-cp
certpassword]\n" );
  printf( "-p   'parameter' - one of:\n" );
  printf( "  XC Server: specify 'host:port:transport', e.g. '127.0.0.1;15910;ssl|tcp'.\n" );
  printf( "  XC Pep   : specify 'comid', e.g. '0'.\n" );
```

```
  printf( "  XC Lite  : specify 'filename', e.g. 'keyexport.dat'.\n" );

  printf( "-u   'user' - User that has the appropriate access rights in the policy.\n" );
  printf( "-d1  'dataelement1' - Data element specified in the policy, to be used in prot,
unprot & reprot\n" );
  printf( "-cu  'username'  - (in case SSL is used) to unlock the client certificate
key.\n" );
  printf( "-cp  'password'  - (in case SSL is used) to unlock the client certificate
key.\n\n" );
  printf( "note: -cu and -cp would be used when:\n" );
  printf( "      when -p is XC Server: '127.0.0.1;15910;ssl'.\n" );
  printf( "      when -p is XC Lite  : 'keyexport.dat'.\n" );
}


/**
 * main program
 */

int main( int argc, char* argv[] )
{
  /* Handle to the XC library */
  XC_HANDLE phXCHandle = XC_NULL;

  /* Handle to the XC session */
  XC_SESSION phSession = XC_NULL;

  /* The parameter can have different values, read xcapi.h */
  XC_CHAR szParameter[512] = {0};

  /* User to decrypt keyexport.dat file */
  XC_CHAR  szUser[256] = {0};

  /* Password to decrypt keyexport.dat file */
  XC_CHAR  szPassword[512] = {0};

  /* User with appropriate priviligies in the DPS policy */
  XC_CHAR szPolicyUser[512]  = {0};

  /* Data element from the DPS policy */
  XC_CHAR szDataElement1[512] = {0};

  /* Data element from the DPS policy */
  XC_CHAR szDataElement2[512] = {0};

  /* Character of data read from the terminal */
  char szBuf[16384] = {0};

  /* Buffer that will hold the protected text of the character */
  /* data read from the terminal */
  XC_BYTE  bOutputData[XC_MAX_OUTPUT_LEN] = {0};

  /* Resulting protected text length */
  XC_UINT4 ui4OutPutDataLength = sizeof( bOutputData );

  /* Buffer that will hold the clear text of the character */
  /* data unprotected from the protected buffer */
  XC_BYTE  bOutputData2[XC_MAX_OUTPUT_LEN] = {0};

  /* Resulting clear text length */
  XC_UINT4 ui4OutPutData2Length = sizeof( bOutputData );

  /* Buffer that will hold the clear text of the character */
  /* data unprotected from the protected buffer */
  XC_BYTE  bOutputDataReprot[XC_MAX_OUTPUT_LEN] = {0};

  /* Resulting clear text length */
  XC_UINT4 ui4OutPutDataReprotLength = sizeof( bOutputData );

  /* Buffer that will hold the unprotected text of the character */
  /* data read from the terminal */
  XC_BYTE  bOutputData3[XC_MAX_OUTPUT_LEN] = {0};
```

```
  /* Resulting unprotected text length */
  XC_UINT4 ui4OutPutData3Length = sizeof( bOutputData3 );

  /* Input data length */
  XC_UINT4 ui4InPutDataLength = 0;

  /* Loop counter */
  XC_UINT4 ui4Count = 0;

  XC_INT4   i4ArgNo = 0;

  /* Version info */
  XC_CHAR  szVersion[512] = {0};

  /* Return Code */
  XC_RETURNCODE rReturnCode = XC_FAILED;

  XC_PARAM_EX stXCParamExProt    = {0};
  XC_PARAM_EX stXCParamExUnprot  = {0};
  XC_PARAM_EX stXCParamExReprot  = {0};
  XC_PARAM_EX stXCParamExUnprot2 = {0};

  stXC_ACTION_RESULT stActionResultProt;
  stXC_ACTION_RESULT stActionResultUnprot;
  stXC_ACTION_RESULT stActionResultReprot;
  stXC_ACTION_RESULT stActionResultUnprot2;

  XC_BYTE bOutNullInd = XC_FALSE;

  /* Parse input arguments. */
  for( i4ArgNo = 1; i4ArgNo < argc; i4ArgNo++ )
  {
    if( 0 == ( strcmp("-p", argv[i4ArgNo] ) ) )
    {
      memcpy( szParameter, argv[i4ArgNo+1], strlen( argv[i4ArgNo+1] ) );
      i4ArgNo++;
    }

    if( 0 == ( strcmp("-cu", argv[i4ArgNo] ) ) )
    {
      memcpy( szUser, argv[i4ArgNo+1], strlen( argv[i4ArgNo+1] ) );
      i4ArgNo++;
    }

    if( 0 == ( strcmp("-cp", argv[i4ArgNo] ) ) )
    {
      memcpy( szPassword, argv[i4ArgNo+1], strlen( argv[i4ArgNo+1] ) );
      i4ArgNo++;
    }

    if( 0 == ( strcmp("-u", argv[i4ArgNo] ) ) )
    {
      memcpy( szPolicyUser, argv[i4ArgNo+1], strlen( argv[i4ArgNo+1] ) );
      i4ArgNo++;
    }

    if( 0 == ( strcmp("-d1", argv[i4ArgNo] ) ) )
    {
      memcpy( szDataElement1, argv[i4ArgNo+1], strlen( argv[i4ArgNo+1] ) );
      i4ArgNo++;
    }

    if( 0 == ( strcmp("-d2", argv[i4ArgNo] ) ) )
    {
      memcpy( szDataElement2, argv[i4ArgNo+1], strlen( argv[i4ArgNo+1] ) );
      i4ArgNo++;
    }

    if( 0 == (strcmp( "-h", argv[i4ArgNo] ) ) )
    {
      printUsage();
      return(0);
    }
```

```
  }

  if( argc < 4 )
  {
    printf("use option '-h' for help\n");
    return(0);
  }

  /**
   * Get version info
   */

  if( XC_SUCCESS == XCGetVersion( szVersion, sizeof( szVersion ) ) )
  {
    printf( "%s\n", szVersion );
  }
  else
  {
    fprintf( stderr, "Failed to get version!\n\n" );
    return -1;
  }

  /**
   * Initialzie the API
   */
  rReturnCode = XCInitLib( &phXCHandle, "" );

  if( XC_SUCCESS ==  rReturnCode )
  {
    /* Read character string from the terminal */
    printf( "Enter a string to protect and press enter: " );
    fgets( szBuf, 16384, stdin );
    printf( "\n" );

    ui4InPutDataLength = strlen( szBuf );
    /* Remove the last input '\n' */
    if( strlen( szBuf ) < 16384 )
    {
      ui4InPutDataLength -= 1;
      szBuf[ui4InPutDataLength] = '\0';
    }

    /**
     * Open session
     */
    printf( "Open session to XC..." );
    rReturnCode = XCOpenSession( phXCHandle,
                                 szUser,
                                 szPassword,
                                 szParameter,
                                 &phSession );


    if( XC_SUCCESS ==  rReturnCode )

    {
      printf( "OK\n\n" );

      printf( "Protecting..." );

      P_strncpy( stXCParamExProt.szVendor, sizeof( stXCParamExProt.szVendor), "XC",
strlen( "XC" ) );
      stXCParamExProt.ui4DataType   = XC_DATATYPE_BYTE;
      stXCParamExProt.ui4Operation  = XC_ANY_FUNCTION;

      rReturnCode = XCProtect( phXCHandle,
                               phSession,
                               XC_TRUE,
                               szPolicyUser,
                               szDataElement1,
                               XC_NULL,
                               0,
                               ( XC_BYTE* )szBuf,
```

```
                                ui4InPutDataLength,
                                XC_FALSE,
                                bOutputData,
                                &ui4OutPutDataLength,
                                &bOutNullInd,
                                &stXCParamExProt,
                                sizeof( stXCParamExProt ),
                                &stActionResultProt );

      /* Protect the characters read from the terminal */
      if( XC_SUCCESS == rReturnCode && XC_LOGRETURNSUCCESS ==
stActionResultProt.ui4LogSeverity )
      {
        printf( "OK\n" );
        printf( "Protected Data: " );

        for( ui4Count = 0; ui4Count < ui4OutPutDataLength; ui4Count++ )
        {
          printf( "%02x", bOutputData[ui4Count] );
        }
        printf( "\n\n" );
        printf( "Unprotecting..." );

        P_strncpy( stXCParamExUnprot.szVendor, sizeof( stXCParamExUnprot.szVendor), "XC",
strlen( "XC" ) );
        stXCParamExUnprot.ui4DataType   = XC_DATATYPE_BYTE;
        stXCParamExUnprot.ui4Operation  = XC_ANY_FUNCTION;

        rReturnCode =  XCUnprotect( phXCHandle,
                                    phSession,
                                    XC_TRUE,
                                    szPolicyUser,
                                    szDataElement1,
                                    XC_NULL,
                                    0,
                                    ( XC_BYTE* )bOutputData,
                                    ui4OutPutDataLength,
                                    XC_FALSE,
                                    bOutputData2,
                                    &ui4OutPutData2Length,
                                    &bOutNullInd,
                                    &stXCParamExUnprot,
                                    sizeof( stXCParamExUnprot ),
                                    &stActionResultUnprot );

        /* Protect the characters read from the terminal */
        if( XC_SUCCESS == rReturnCode && XC_LOGRETURNSUCCESS ==
stActionResultUnprot.ui4LogSeverity )
        {
          printf( "OK\n" );
          printf( "Clear text: " );

          /* Null terminate */
          bOutputData2[ui4OutPutData2Length] = '\0';
          printf( "%s\n\n", bOutputData2 );
        }
        else
        {
          printf( "\nFailed to Unprotect:error code:%d\n", rReturnCode );
        }
      }
      else
      {
        printf( "\nFailed to protect:error code:%d\n", rReturnCode );
      }

      if( XC_SUCCESS ==  rReturnCode )
      {
        printf( "OK\n" );
        printf( "Reprotecting..." );

        stXCParamExUnprot.ui4DataType   = XC_DATATYPE_BYTE;
        stXCParamExUnprot2.ui4Operation  = XC_ANY_FUNCTION;
```

```
        P_strncpy( stXCParamExUnprot.szVendor, sizeof( stXCParamExUnprot.szVendor), "XC",
strlen( "XC" ) );

        rReturnCode = XCReprotect( phXCHandle,
                                   phSession,
                                   XC_TRUE,
                                   szPolicyUser,
                                   szDataElement1,
                                   szDataElement2,
                                   XC_NULL,
                                   0,
                                   XC_NULL,
                                   0,
                                   ( XC_BYTE* )bOutputData,
                                   ui4OutPutDataLength,
                                   XC_FALSE,
                                   bOutputDataReprot,
                                   &ui4OutPutDataReprotLength,
                                   &bOutNullInd,
                                   &stXCParamExReprot,
                                   sizeof( stXCParamExReprot ),
                                   &stActionResultReprot );

        if( XC_SUCCESS == rReturnCode && XC_LOGRETURNSUCCESS ==
stActionResultReprot.ui4LogSeverity )
        {
          printf( "Reprotected Data: " );
          for( ui4Count = 0; ui4Count < ui4OutPutDataReprotLength; ui4Count++ )
          {
            printf( "%02x", bOutputDataReprot[ui4Count] );
          }
          printf( "\n\n" );
        }
        else
        {
          printf( "\nFailed to Reprotect:error code:%d\n", rReturnCode );
        }

        printf( "Unprotecting..." );

        P_strncpy( stXCParamExUnprot2.szVendor, sizeof( stXCParamExUnprot2.szVendor), "XC",
strlen( "XC" ) );
        stXCParamExUnprot2.ui4DataType   = XC_DATATYPE_BYTE;
        stXCParamExUnprot2.ui4Operation  = XC_ANY_FUNCTION;

        rReturnCode =  XCUnprotect( phXCHandle,
                                    phSession,
                                    XC_TRUE,
                                    szPolicyUser,
                                    szDataElement2,
                                    XC_NULL,
                                    0,
                                    ( XC_BYTE* )bOutputDataReprot,
                                    ui4OutPutDataReprotLength,
                                    XC_FALSE,
                                    bOutputData3,
                                    &ui4OutPutData3Length,
                                    &bOutNullInd,
                                    &stXCParamExUnprot2,
                                    sizeof( stXCParamExUnprot2 ),
                                    &stActionResultUnprot2 );

        /* Protect the characters read from the terminal */
        if( XC_SUCCESS == rReturnCode && XC_LOGRETURNSUCCESS ==
stActionResultUnprot2.ui4LogSeverity )
        {
          printf( "OK\n" );
          printf( "Clear text: " );

          /* Null terminate */
          bOutputData3[ui4OutPutData3Length] = '\0';
          printf( "%s\n\n", bOutputData3 );
        }
```

```
        else
        {
          printf( "\nFailed to Unprotect:error code:%d\n", rReturnCode );
        }
      }
    }
    else
    {
      printf( "\nFailed to open session:error code:%d\n", rReturnCode );
    }

    XCCloseSession( phXCHandle, &phSession );

  }
  else
  {
    printf( "Failed to initialize XC API, error code:%d\n", rReturnCode );
  }

  XCTerminateLib( &phXCHandle );

  return 0;
}
```

9.7.1.2 Using the AP C APIs on Windows

This section describes how to use the AP C APIs on a Windows platform using a sample application.

After you have setup the the policy, you can begin testing the AP C APIs to perform protect, unprotect, and reprotect operations.

To run the sample code, perform the following steps.

1. Open the x64 Native Tools Command Prompt provided by the Visual Studio application.
2. Set the path of the *xcpep.plm* file to the PATH system variable.

   For example,

   ```
   set PATH=%PATH%;C:\Program Files\Protegrity\Defiance XC\bin
   ```

3. Run the following command to compile the sample code.

   ```
   cl /I "C:\Program Files\Protegrity\Defiance XC\include" /D "_WIN64" /MT "C:\Program Files\Protegrity\Defiance XC\lib\xcpep.lib" sample.c
   ```

   A *sample.exe* executable file is created.

4. Run the following sample command to perform the protect, unprotect, and reprotect operations.

   ```
   sample.exe -p parameter -u user -d1 dataelement1 [-d2 dataelement2] (-prot | -unprot | -reprot | -hmac | -check | -currkid dataelement | -getkid dataelement data) [-in=hex|raw|int] [-out=text|raw|int] [-op <operation>] [-cu certuser] [-cp certpassword] [-dir workingdir] [-batch records] [-version] [-defde policy] [-data data | NULL] [-scid seccoordID] [-f datafile]
   ```

> **Note:**
> If you want to see a list of available parameters, then run the following command.
>
> ```
> sample.exe -h
> ```

> **Caution:**

When a short running application has completed its execution (in less than a second), the audit logs will not be seen. In such cases, the *XCFlushPepAudits* function() API needs to be invoked.

For more information about the *XCFlushPepAudits* function, refer to the section *XCFlushPepAudits Function* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

For more information on the AP C methods, refer to section *Application Protector (AP) C APIs* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.0*.

The following is a sample application for the AP C.

```
/*
 * AP-C Sample code to protect some data.
 * Refer readme for compile and execute instructions.
 */

#include <stdio.h>
#include <string.h>

/* xcapi.h includes the AP-C API signature.
 * xcdefinitions.h includes limitations and structures.
 **/
#include "xcapi.h"

/**
 * Maximum growth factor for
 * Tokenization : 1.35 ( eg: Unicode Token )
 * FPE          : 2.00 ( eg: UTF-16 BE/LE  )
 * Encryption   : 512  ( eg: AES 128/256 and etc. )
 */

#define XC_MAX_INPUT_LEN  ((4*(4096+2)) + XC_BYTES_OVERHEAD )
#define XC_MAX_OUTPUT_LEN ((2*(XC_MAX_INPUT_LEN+1)) + XC_BYTES_OVERHEAD )


void printUsage()
{
  printf( "Usage: xcenc -p parameter -u user -d1 dataelement1 [-cu certuser] [-cp
certpassword]\n" );
  printf( "-p   'parameter' - one of:\n" );
  printf( "  XC Server: specify 'host:port:transport', e.g. '127.0.0.1;15910;ssl|tcp'.\n" );
  printf( "  XC Pep   : specify 'comid', e.g. '0'.\n" );
  printf( "  XC Lite  : specify 'filename', e.g. 'keyexport.dat'.\n" );

  printf( "-u   'user' - User that has the appropriate access rights in the policy.\n" );
  printf( "-d1  'dataelement1' - Data element specified in the policy, to be used in prot,
unprot & reprot\n" );
  printf( "-cu  'username'  - (in case SSL is used) to unlock the client certificate
key.\n" );
  printf( "-cp  'password'  - (in case SSL is used) to unlock the client certificate
key.\n\n" );
  printf( "note: -cu and -cp would be used when:\n" );
  printf( "      when -p is XC Server: '127.0.0.1;15910;ssl'.\n" );
  printf( "      when -p is XC Lite  : 'keyexport.dat'.\n" );
}


/**
 * main program
 */

int main( int argc, char* argv[] )
{
  /* Handle to the XC library */
  XC_HANDLE phXCHandle = XC_NULL;

  /* Handle to the XC session */
  XC_SESSION phSession = XC_NULL;
```

```
/* The parameter can have different values, read xcapi.h */
XC_CHAR szParameter[512] = {0};

/* User to decrypt keyexport.dat file */
XC_CHAR  szUser[256] = {0};

/* Password to decrypt keyexport.dat file */
XC_CHAR  szPassword[512] = {0};

/* User with appropriate priviligies in the DPS policy */
XC_CHAR szPolicyUser[512]  = {0};

/* Data element from the DPS policy */
XC_CHAR szDataElement1[512] = {0};

/* Data element from the DPS policy */
XC_CHAR szDataElement2[512] = {0};

/* Character of data read from the terminal */
char szBuf[16384] = {0};

/* Buffer that will hold the protected text of the character */
/* data read from the terminal */
XC_BYTE  bOutputData[XC_MAX_OUTPUT_LEN] = {0};

/* Resulting protected text length */
XC_UINT4 ui4OutPutDataLength = sizeof( bOutputData );

/* Buffer that will hold the clear text of the character */
/* data unprotected from the protected buffer */
XC_BYTE  bOutputData2[XC_MAX_OUTPUT_LEN] = {0};

/* Resulting clear text length */
XC_UINT4 ui4OutPutData2Length = sizeof( bOutputData );

/* Buffer that will hold the clear text of the character */
/* data unprotected from the protected buffer */
XC_BYTE  bOutputDataReprot[XC_MAX_OUTPUT_LEN] = {0};

/* Resulting clear text length */
XC_UINT4 ui4OutPutDataReprotLength = sizeof( bOutputData );

/* Buffer that will hold the unprotected text of the character */
/* data read from the terminal */
XC_BYTE  bOutputData3[XC_MAX_OUTPUT_LEN] = {0};

/* Resulting unprotected text length */
XC_UINT4 ui4OutPutData3Length = sizeof( bOutputData3 );

/* Input data length */
XC_UINT4 ui4InPutDataLength = 0;

/* Loop counter */
XC_UINT4 ui4Count = 0;

XC_INT4   i4ArgNo = 0;

/* Version info */
XC_CHAR  szVersion[512] = {0};

/* Return Code */
XC_RETURNCODE rReturnCode = XC_FAILED;

XC_PARAM_EX stXCParamExProt    = {0};
XC_PARAM_EX stXCParamExUnprot  = {0};
XC_PARAM_EX stXCParamExReprot  = {0};
XC_PARAM_EX stXCParamExUnprot2 = {0};

stXC_ACTION_RESULT stActionResultProt;
stXC_ACTION_RESULT stActionResultUnprot;
stXC_ACTION_RESULT stActionResultReprot;
stXC_ACTION_RESULT stActionResultUnprot2;
```

```
XC_BYTE bOutNullInd = XC_FALSE;

/* Parse input arguments. */
for( i4ArgNo = 1; i4ArgNo < argc; i4ArgNo++ )
{
  if( 0 == ( strcmp("-p", argv[i4ArgNo] ) ) )
  {
    memcpy( szParameter, argv[i4ArgNo+1], strlen( argv[i4ArgNo+1] ) );
    i4ArgNo++;
  }

  if( 0 == ( strcmp("-cu", argv[i4ArgNo] ) ) )
  {
    memcpy( szUser, argv[i4ArgNo+1], strlen( argv[i4ArgNo+1] ) );
    i4ArgNo++;
  }

  if( 0 == ( strcmp("-cp", argv[i4ArgNo] ) ) )
  {
    memcpy( szPassword, argv[i4ArgNo+1], strlen( argv[i4ArgNo+1] ) );
    i4ArgNo++;
  }

  if( 0 == ( strcmp("-u", argv[i4ArgNo] ) ) )
  {
    memcpy( szPolicyUser, argv[i4ArgNo+1], strlen( argv[i4ArgNo+1] ) );
    i4ArgNo++;
  }

  if( 0 == ( strcmp("-d1", argv[i4ArgNo] ) ) )
  {
    memcpy( szDataElement1, argv[i4ArgNo+1], strlen( argv[i4ArgNo+1] ) );
    i4ArgNo++;
  }

  if( 0 == ( strcmp("-d2", argv[i4ArgNo] ) ) )
  {
    memcpy( szDataElement2, argv[i4ArgNo+1], strlen( argv[i4ArgNo+1] ) );
    i4ArgNo++;
  }

  if( 0 == (strcmp( "-h", argv[i4ArgNo] ) ) )
  {
    printUsage();
    return(0);
  }
}

if( argc < 4 )
{
  printf("use option '-h' for help\n");
  return(0);
}

/**
 * Get version info
 */

if( XC_SUCCESS == XCGetVersion( szVersion, sizeof( szVersion ) ) )
{
  printf( "%s\n", szVersion );
}
else
{
  fprintf( stderr, "Failed to get version!\n\n" );
  return -1;
}

/**
 * Initialzie the API
 */
rReturnCode = XCInitLib( &phXCHandle, "" );
```

```
  if( XC_SUCCESS ==  rReturnCode )
  {
    /* Read character string from the terminal */
    printf( "Enter a string to protect and press enter: " );
    fgets( szBuf, 16384, stdin );
    printf( "\n" );

    ui4InPutDataLength = strlen( szBuf );
    /* Remove the last input '\n' */
    if( strlen( szBuf ) < 16384 )
    {
      ui4InPutDataLength -= 1;
      szBuf[ui4InPutDataLength] = '\0';
    }

    /**
     * Open session
     */
    printf( "Open session to XC..." );
    rReturnCode = XCOpenSession( phXCHandle,
                                 szUser,
                                 szPassword,
                                 szParameter,
                                 &phSession );


    if( XC_SUCCESS ==  rReturnCode )

    {
      printf( "OK\n\n" );

      printf( "Protecting..." );

      P_strncpy( stXCParamExProt.szVendor, sizeof( stXCParamExProt.szVendor), "XC",
strlen( "XC" ) );
      stXCParamExProt.ui4DataType   = XC_DATATYPE_BYTE;
      stXCParamExProt.ui4Operation  = XC_ANY_FUNCTION;

      rReturnCode = XCProtect( phXCHandle,
                               phSession,
                               XC_TRUE,
                               szPolicyUser,
                               szDataElement1,
                               XC_NULL,
                               0,
                               ( XC_BYTE* )szBuf,
                               ui4InPutDataLength,
                               XC_FALSE,
                               bOutputData,
                               &ui4OutPutDataLength,
                               &bOutNullInd,
                               &stXCParamExProt,
                               sizeof( stXCParamExProt ),
                               &stActionResultProt );

      /* Protect the characters read from the terminal */
      if( XC_SUCCESS == rReturnCode && XC_LOGRETURNSUCCESS ==
stActionResultProt.ui4LogSeverity )
      {
        printf( "OK\n" );
        printf( "Protected Data: " );

        for( ui4Count = 0; ui4Count < ui4OutPutDataLength; ui4Count++ )
        {
          printf( "%02x", bOutputData[ui4Count] );
        }
        printf( "\n\n" );
        printf( "Unprotecting..." );

        P_strncpy( stXCParamExUnprot.szVendor, sizeof( stXCParamExUnprot.szVendor), "XC",
strlen( "XC" ) );
        stXCParamExUnprot.ui4DataType   = XC_DATATYPE_BYTE;
```

```
        stXCParamExUnprot.ui4Operation  = XC_ANY_FUNCTION;

        rReturnCode =  XCUnprotect( phXCHandle,
                                    phSession,
                                    XC_TRUE,
                                    szPolicyUser,
                                    szDataElement1,
                                    XC_NULL,
                                    0,
                                    ( XC_BYTE* )bOutputData,
                                    ui4OutPutDataLength,
                                    XC_FALSE,
                                    bOutputData2,
                                    &ui4OutPutData2Length,
                                    &bOutNullInd,
                                    &stXCParamExUnprot,
                                    sizeof( stXCParamExUnprot ),
                                    &stActionResultUnprot );

        /* Protect the characters read from the terminal */
        if( XC_SUCCESS == rReturnCode && XC_LOGRETURNSUCCESS ==
stActionResultUnprot.ui4LogSeverity )
        {
          printf( "OK\n" );
          printf( "Clear text: " );

          /* Null terminate */
          bOutputData2[ui4OutPutData2Length] = '\0';
          printf( "%s\n\n", bOutputData2 );
        }
        else
        {
          printf( "\nFailed to Unprotect:error code:%d\n", rReturnCode );
        }
      }
      else
      {
        printf( "\nFailed to protect:error code:%d\n", rReturnCode );
      }

      if( XC_SUCCESS ==  rReturnCode )
      {
        printf( "OK\n" );
        printf( "Reprotecting..." );

        stXCParamExUnprot.ui4DataType   = XC_DATATYPE_BYTE;
        stXCParamExUnprot2.ui4Operation  = XC_ANY_FUNCTION;
        P_strncpy( stXCParamExUnprot.szVendor, sizeof( stXCParamExUnprot.szVendor), "XC",
strlen( "XC" ) );

        rReturnCode = XCReprotect( phXCHandle,
                                   phSession,
                                   XC_TRUE,
                                   szPolicyUser,
                                   szDataElement1,
                                   szDataElement2,
                                   XC_NULL,
                                   0,
                                   XC_NULL,
                                   0,
                                   ( XC_BYTE* )bOutputData,
                                   ui4OutPutDataLength,
                                   XC_FALSE,
                                   bOutputDataReprot,
                                   &ui4OutPutDataReprotLength,
                                   &bOutNullInd,
                                   &stXCParamExReprot,
                                   sizeof( stXCParamExReprot ),
                                   &stActionResultReprot );

        if( XC_SUCCESS == rReturnCode && XC_LOGRETURNSUCCESS ==
stActionResultReprot.ui4LogSeverity )
        {
```

```
          printf( "Reprotected Data: " );
          for( ui4Count = 0; ui4Count < ui4OutPutDataReprotLength; ui4Count++ )
          {
            printf( "%02x", bOutputDataReprot[ui4Count] );
          }
          printf( "\n\n" );
        }
        else
        {
          printf( "\nFailed to Reprotect:error code:%d\n", rReturnCode );
        }

        printf( "Unprotecting..." );

        P_strncpy( stXCParamExUnprot2.szVendor, sizeof( stXCParamExUnprot2.szVendor), "XC",
strlen( "XC" ) );
        stXCParamExUnprot2.ui4DataType   = XC_DATATYPE_BYTE;
        stXCParamExUnprot2.ui4Operation  = XC_ANY_FUNCTION;

        rReturnCode =  XCUnprotect( phXCHandle,
                                    phSession,
                                    XC_TRUE,
                                    szPolicyUser,
                                    szDataElement2,
                                    XC_NULL,
                                    0,
                                    ( XC_BYTE* )bOutputDataReprot,
                                    ui4OutPutDataReprotLength,
                                    XC_FALSE,
                                    bOutputData3,
                                    &ui4OutPutData3Length,
                                    &bOutNullInd,
                                    &stXCParamExUnprot2,
                                    sizeof( stXCParamExUnprot2 ),
                                    &stActionResultUnprot2 );

        /* Protect the characters read from the terminal */
        if( XC_SUCCESS == rReturnCode && XC_LOGRETURNSUCCESS ==
stActionResultUnprot2.ui4LogSeverity )
        {
          printf( "OK\n" );
          printf( "Clear text: " );

          /* Null terminate */
          bOutputData3[ui4OutPutData3Length] = '\0';
          printf( "%s\n\n", bOutputData3 );
        }
        else
        {
          printf( "\nFailed to Unprotect:error code:%d\n", rReturnCode );
        }
      }
    }
    else
    {
      printf( "\nFailed to open session:error code:%d\n", rReturnCode );
    }

    XCCloseSession( phXCHandle, &phSession );

  }
  else
  {
    printf( "Failed to initialize XC API, error code:%d\n", rReturnCode );
  }

  XCTerminateLib( &phXCHandle );

  return 0;
}
```

9.7.2 Using The AP Java APIs

This section describes how to use the AP Java APIs using a sample application.

After you have setup the the policy and trusted application, you can begin testing the AP Java APIs for protection, unprotection, and reprotection.

> **Caution:**
> When a short running application has completed its execution (in less than a second), the audit logs will not be seen. In such cases, the *protector.flushAudits()* API needs to be invoked.
>
> For more information on flushAudits, refer to the section *flushAudits* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

For more information on the AP Java APIs, refer to section *Application Protector (AP) Java APIs* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

For more information about the AP Java error return codes, refer to the section *Application Protectors API Return Codes* in the *Protegrity Troubleshooting Guide 9.1.0.4*.

The following text represents a sample code for the protection and unprotection operations using the AP Java APIs.

```
/* HelloWorld.java
*
* Illustrates how to call the new java api
* Executing this for the first time will create a forensic entry that will
* have to be added to the authorized app
* Compiled as  : javac -cp ApplicationProtectorJava.jar HelloWorld.java
* Run as       :
* java -cp  ApplicationProtectorJava.jar HelloWorld AuthPolicyUser DataElement Data
* java version : 1.8.0_45
* Package Name: ApplicationProtector_Linux-ALL-64_x86-64_JRE-1.8-64_9.1.0.0.43.tgz
*
* Use either token elements or DTP2 elements or NoEncryption as DataElement
* while running this code.
*/

import com.protegrity.ap.java.*;

public class HelloWorld {
    public static void main(String[] args) throws ProtectorException {
        if (args.length == 3) {
            System.out.println(" AuthUser : "+args[0]);
            System.out.println(" DataElement : "+args[1]);
            System.out.println(" Clear Text Data : "+args[2]);
            stringTest(args[0],args[1],args[2]);
        } else {
            System.out.println(" Usage : java -cp ApplicationProtectorJava.jar HelloWorld
AuthUser DataElement Data");
            System.out.println(" Example : java -cp ApplicationProtectorJava.jar HelloWorld
USER TKCCN 4111111111111111");
            System.exit(0);
        }
    }

    static public void stringTest(String USR, String DE, String Data) throws
ProtectorException {
        boolean result = false;
        String[] inputStringArray = new String[1];
        String[] ProtectStringArray = new String[1];
        String[] ReProtectStringArray = new String[1];
        String[] UnProtectStringArray = new String[1];
        SessionObject session = null;
        Protector protector = null;
```

```
        inputStringArray[0] = Data;

        try {
            /** Instantiate the protector
             * This should be invoked only once in the lifetime of each application that uses
the protector.
             * Protection operations can be performed only on successful initialization of
the protector.
            **/
            protector = Protector.getProtector();
            System.out.println("1.) Protector Instantiated!! ");

            // Set input parameters and create session
            session = protector.createSession(USR);
            System.out.println("2.) Session created ");

            // test protect/reprotect/unprotect methods
            if (!protector.protect(session, DE, inputStringArray, ProtectStringArray))
                System.out.println("protect api failed !!! \nError : " +
protector.getLastError(session));
            else
                System.out.println("3.) Protect Output   : " + ProtectStringArray[0]);

            /*
             * if(!protector.reprotect(session,DE,DE,ProtectStringArray,ReProtectStringArray
             * ))
             * System.out.println("reprotect api failed !!! \nError : "+protector.
             * getLastError(session));
             * else
             * System.out.println( "4.) ReProtect Output: " + ReProtectStringArray[0] );
             * if(!protector.unprotect(session,DE,ReProtectStringArray,UnProtectStringArray)
             * )
             * System.out.println("unprotect api failed !!! \nError : "+protector.
             * getLastError(session));
             * else
             * System.out.println( "4.) UnProtect Output: " + UnProtectStringArray[0] );
             */
            if (!protector.unprotect(session, DE, ProtectStringArray, UnProtectStringArray))
                System.out.println("unprotect api failed !!! \nError : " +
protector.getLastError(session));
            else
                System.out.println("4.) UnProtect Output : " + UnProtectStringArray[0]);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // FlushAudits should be invoked only once at the end of the application
lifecycle.
            protector.flushAudits();
        }
    }
}
```

9.7.3 Using The AP Python APIs

After you have set up the the policy and Trusted Application, you can begin testing AP Python APIs for protection, unprotection, and reprotection. The following code snippet provides an example for protecting, unprotecting, and reprotecting string data.

For more information about the AP Python error return codes, refer to the section *Application Protectors API Return Codes* in the *Protegrity Troubleshooting Guide 9.1.0.4*.

```
# -*- coding: utf-8 -*-

from appython import Protector


if __name__ == "__main__":

    # Initialize the protector
    protector = Protector()
```

```
    # Create session with policy user
    session = protector.create_session("USER1")

    # Protect operation
    p_out = session.protect("Protegrity1", "TE_A_N_S23_L2R2_Y")
    print("Protected Data: %s" %p_out)

    # Reprotect operation
    r_out = session.reprotect(p_out, "TE_A_N_S23_L2R2_Y", "TE_A_N_S23_L2R2_Y")
    print("Reprotected Data: %s" %r_out)

    # Unprotect operation
    org = session.unprotect(r_out, "TE_A_N_S23_L2R2_Y")
    print("Unprotected Data: %s" %org)
```

9.7.4 Using the AP Go APIs

> **Note:**
> User must call *defer session.FlushAudits()* method after creating the session to see the logs for smaller running applications.

After you have set up the policy and trusted application, you can begin testing AP Go APIs for protection and unprotection.

Navigate to the ***/opt/protegrity/applicationprotector/go/src/protegrity.com/samples/*** directory and run the following command to launch the sample *Hello World* application for AP Go.

```
go run hello_world.go inputdata authPolicyUser dataElement newDataElement externalIv
newExternalIv
```

The following code snippet is from the sample *hello_world* application.

For more information about the AP Go error return codes, refer to the section *Application Protectors API Return Codes* in the *Protegrity Troubleshooting Guide 9.1.0.4*.

> **Note:**
> Ensure that the following Go environment variable is set to *off*.
>
> ***go env -w GO111MODULE="off"***

```
/*This is sample program demonstrating the usage of protegrity.com/apgo APIs.

* Configure Trusted Application policy in ESA with
    - Application name: hello_world
    - Application user: <SYSTEM USER>
* Ensure that the following environment variables have been set,
  assuming AP Go has been installed in '/opt/protegrity/':
    - export GOPATH=$GOPATH:/opt/protegrity/applicationprotector/go/
    - export CGO_CFLAGS="-I/opt/protegrity/applicationprotector/go/include"
    - export CGO_LDFLAGS="-L/opt/protegrity/applicationprotector/go/lib"
    - export LD_LIBRARY_PATH=/opt/protegrity/applicationprotector/go/lib
* Browse to the directory /opt/protegrity/applicationprotector/go/src/protegrity.com/samples/
    - go run hello_world.go inputdata authPolicyUser dataElement newDataElement externalIv
newExternalIv
*/
package main

import (
    "fmt"
    "os"
```

```
    apgo "protegrity.com/apgo"
)

func main() {

    if len(os.Args) == 7 {

        input := os.Args[1]
        policyUsr := os.Args[2]
        dataElement := os.Args[3]
        newDataElement := os.Args[4]
        externalIv := os.Args[5]
        newExternalIv := os.Args[6]

        fmt.Printf("Testing with new data element = '%v' data element - '%v' policy user -
'%v' external iv - '%v' new external iv - '%v'\n", newDataElement, dataElement, policyUsr,
externalIv, newExternalIv)

        fmt.Printf("AP Go Version: %s\n\n", apgo.GetVersion())

        // initilize the AP Go Package
        terminate, err := apgo.Init()
        if err != nil {
            panic(fmt.Sprintf("apgo.InitLib() failed. Error: %v", err))
        }
        defer terminate()
        fmt.Print("Input Data: ", input, "\n")
        // create a new Protection Operation session for policyUsr
        session, err := apgo.NewSession(policyUsr)
        if err != nil {
            panic(fmt.Sprintf("apgo.NewSession() failed. Error: %v", err))
        }

        // Perform Single Protect/Unprotect/Reprotect Operations

        fmt.Print("Performing Single Protect/Unprotect/Reprotect Operations \n")

        output, rc, err := session.ProtectStr(input, dataElement)
        if err != nil {
            panic(fmt.Sprintf("session.ProtectStr() failed. Return Code: %v, Error: %v", rc,
err))
        }
        fmt.Printf("Protected Data: %v\n", output)

        org, rc, err := session.UnprotectStr(output, dataElement)
        if err != nil {
            panic(fmt.Sprintf("session.UnprotectStr() failed. Return Code: %v, Error: %v",
rc, err))
        }
        fmt.Printf("Unprotected Data: %v\n", org)

        reoutput, rc, err := session.ReprotectStr(output, dataElement, newDataElement)
        if err != nil {
            panic(fmt.Sprintf("session.ReprotectStr() failed. Return Code: %v, Error: %v",
rc, err))
        }
        fmt.Printf("Reprotected Data: %v\n\n", reoutput)

        //Example of using external iv to perform single Protect/Unprotect/Reprotect
Operations

        fmt.Print("Performing Single Protect/Unprotect/Reprotect Operations using external iv
\n")

        outputexiv, rc, err := session.ProtectStr(input, dataElement,
apgo.UsingExtIV([]byte(externalIv)))
        if err != nil {
            panic(fmt.Sprintf("session.ProtectStr() failed. Return Code: %v, Error: %v", rc,
err))
        }
        fmt.Printf("Protected External IV Data: %v\n", outputexiv)

        orgexiv, rc, err := session.UnprotectStr(outputexiv, dataElement,
```

```
apgo.UsingExtIV([]byte(externalIv)))
        if err != nil {
            panic(fmt.Sprintf("session.UnprotectStr() failed. Return Code: %v, Error: %v",
rc, err))
        }
        fmt.Printf("Unprotected External IV Data: %v\n", orgexiv)

        reoutputiv, rc, err := session.ReprotectStr(outputexiv, dataElement, newDataElement,
apgo.UsingExtIV([]byte(externalIv)), apgo.UsingNewExtIV([]byte(newExternalIv)))
        if err != nil {
            panic(fmt.Sprintf("session.ReprotectStr() failed. Return Code: %v, Error: %v",
rc, err))
        }
        fmt.Printf("Reprotected External IV Data: %v\n\n", reoutputiv)

        blkInput := []string{input, input, input}

        // Perform Bulk Protect/Unprotect/Reprotect Operations

        fmt.Print("Performing Bulk Protect/Unprotect/Reprotect \n")

        blkOut, retCodes, err := session.ProtectStrBulk(blkInput, dataElement)
        if err != nil {
            panic(fmt.Sprintf("session.ProtectStrBulk() failed. Return Codes: %v, Error: %v",
retCodes, err))
        }
        fmt.Printf("Protected Bulk Data: %v, len: %v\n", blkOut, len(blkOut))

        blkOrg, retCodes, err := session.UnprotectStrBulk(blkOut, dataElement)
        if err != nil {
            panic(fmt.Sprintf("session.UnprotectStrBulk() failed. Return Codes: %v, Error:
%v", retCodes, err))
        }
        fmt.Printf("Original Bulk Data: %v, len: %v\n", blkOrg, len(blkOrg))

        reblkOut, retCodes, err := session.ReProtectStrBulk(blkOut, dataElement,
newDataElement)
        if err != nil {
            panic(fmt.Sprintf("session.ReProtectStrBulk() failed. Return Codes: %v, Error:
%v", retCodes, err))
        }
        fmt.Printf("ReProtected Bulk Data: %v, len: %v\n\n", reblkOut, len(reblkOut))

        //Example of using external iv to perform bulk Protect/Unprotect/Reprotect Operations

        fmt.Print("Performing Bulk Protect/Unprotect/Reprotect Operations using external iv
\n")

        blkOuteiv, retCodes, err := session.ProtectStrBulk(blkInput, dataElement,
apgo.UsingExtIV([]byte(externalIv)))
        if err != nil {
            panic(fmt.Sprintf("session.ProtectStrBulk() failed. Return Codes: %v, Error: %v",
retCodes, err))
        }
        fmt.Printf("Protected External IV Bulk Data: %v, len: %v\n", blkOuteiv,
len(blkOuteiv))

        blkOrgeiv, retCodes, err := session.UnprotectStrBulk(blkOuteiv, dataElement,
apgo.UsingExtIV([]byte(externalIv)))
        if err != nil {
            panic(fmt.Sprintf("session.UnprotectStrBulk() failed. Return Codes: %v, Error:
%v", retCodes, err))
        }
        fmt.Printf("Original External IV Bulk Data: %v, len: %v\n", blkOrgeiv, len(blkOrgeiv))

        reblkOuteiv, retCodes, err := session.ReProtectStrBulk(blkOut,
dataElement, newDataElement, apgo.UsingExtIV([]byte(externalIv)),
apgo.UsingNewExtIV([]byte(newExternalIv)))
        if err != nil {
            panic(fmt.Sprintf("session.ReProtectStrBulk() failed. Return Codes: %v, Error:
%v", retCodes, err))
        }
        fmt.Printf("ReProtected External IV Bulk Data: %v, len: %v\n", reblkOuteiv,
```

```
len(reblkOuteiv))
    } else {
        fmt.Print("Required Arguments are not provided\n")
        fmt.Print("Usage   : go run hello_world.go inputdata authPolicyUser dataElement
newDataElement externalIv newExternalIv \n")
        fmt.Print("Example : go run hello_world.go HelloWorld user1 Alphanumeric
Alphanumeric1  Protegrity Protegrity$$ \n")
    }

}
```

9.7.5 Using the AP NodeJS APIs

This section describes how to use the AP NodeJS APIs using a sample application.

After you have setup the policy and trusted application, you can begin testing the AP NodeJS APIs for protect, unprotect, and reprotect operations.

> **Note:** To run this sample application, ensure that the Application Name in the Trusted Application is set as *helloworld*.

For more information on the AP NodeJS APIs, refer to section *Application Protector (AP) NodeJS APIs* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.0*.

For more information about the AP NodeJS error return codes, refer to the section *Application Protectors API Return Codes* in the *Protegrity Troubleshooting Guide 9.1.0.4*.

> **Caution:**
> When a short running application has completed its execution (in less than a second), the audit logs will not be seen. In such cases, the *flushAudits()* API needs to be invoked.
>
> For more information about the flushAudits API, refer to the section *flushAudits API* in the *Protegrity APIs, UDFs, and Commands Reference Guide 9.1.0.0*.

To run the AP NodeJS sample:

1. Navigate to the directory where you have saved the sample *helloworld* application for AP NodeJS.
2. Run the following command to launch the sample *helloworld* application for AP NodeJS.

```
node helloworld.js inputdata authPolicyUser dataElement newDataElement externalIV
newExternalIV
```

The following code snippet provides an example for protecting, unprotecting, and reprotecting the string data.

```
const protector = require('apnode')

if (process.argv.length == 8) {
     const input = process.argv[2]
     const user = process.argv[3]
     const dataElement = process.argv[4]
     const newDataElement = process.argv[5]
     const externalIV = process.argv[6]
     const newExternalIV = process.argv[7]

     protector.initialize().then(result => {
        if (result) {

            console.log("\n"+"Testing with newDataElement -", newDataElement," dataElement
-", dataElement ," policyUser -" , user, " externalIV -", externalIV," newExternalIV -" ,
newExternalIV +"\n")
```

```
            //getVersion

            console.log("AP Node Version:",protector.getVersion(), "\n")

            // Perform Single Protect/Unprotect/Reprotect Operations

            const protectedData = protector.protect(input, user, dataElement)
            console.log("Protected Data", protectedData)

            const unprotectedData =  protector.unprotect(protectedData.output, user,
dataElement)
            console.log("Unprotected Data", unprotectedData)

            const reprotectedData =  protector.reprotect(protectedData.output, user,
dataElement, newDataElement)
            console.log("Reprotected Data", reprotectedData)

            console.log("\n")

            //Example of using external iv to perform single Protect/Unprotect/Reprotect
Operations

            const protectedDataIV = protector.protect(input, user, dataElement,
Buffer.from(externalIV))
            console.log("Protected External IV Data", protectedDataIV)

            const unprotectedDataIV =  protector.unprotect(protectedDataIV.output, user,
dataElement, Buffer.from(externalIV))
            console.log("Unprotected External IV Data", unprotectedDataIV)

            const reprotectedDataIV =  protector.reprotect(protectedDataIV.output, user,
dataElement, newDataElement, Buffer.from(externalIV), Buffer.from(newExternalIV))
            console.log("Reprotected External IV Data", reprotectedDataIV)

            console.log("\n")

            blkInput = [input, input, input]

            // Perform Bulk Protect/Unprotect/Reprotect Operations

            const bulkprotectedData = protector.protect(blkInput, user, dataElement)
            console.log("Bulk Protected Data", bulkprotectedData)

            const bulkunprotectedData =  protector.unprotect(bulkprotectedData.output, user,
dataElement)
            console.log("Bulk Unprotected Data", bulkunprotectedData)

            const bulkreprotectedData =  protector.reprotect(bulkprotectedData.output, user,
dataElement, newDataElement)
            console.log("Bulk Reprotected Data", bulkreprotectedData)

            console.log("\n")

            //Example of using external iv to perform bulk Protect/Unprotect/Reprotect
Operations

            const bulkprotectedDataIV = protector.protect(blkInput, user, dataElement,
Buffer.from(externalIV) )
            console.log("Bulk Protected External IV Data", bulkprotectedDataIV)

            const bulkunprotectedDataIV =  protector.unprotect(bulkprotectedDataIV.output,
user, dataElement, Buffer.from(externalIV))
            console.log("Bulk Unprotected External IV Data", bulkunprotectedDataIV)

            const bulkreprotectedDataIV =  protector.reprotect(bulkprotectedDataIV.output,
user, dataElement, newDataElement, Buffer.from(externalIV), Buffer.from(newExternalIV))
            console.log("Bulk Reprotected External IV Data", bulkreprotectedDataIV)
        }

    }).catch(error => {
        //handle exceptions
        console.error(error)
    }).finally(() => {
```

```
        //flush audits
        protector.flushAudits()
    })
}else {
    console.log("Required Arguments are not provided\n")
    console.log("Usage   : node helloworld.js inputdata authPolicyUser dataElement
newDataElement externalIV newExternalIV \n")
    console.log("Example : node helloworld.js HelloWorld user1 TE_A_N_S13_L1R3_N
TE_A_N_S13_L1R3_N1 protegrity protegrity123 \n")
}
```

9.7.6 Using The AP .NET APIs

This section describes how to use the AP .NET APIs using a sample application.

After you have setup the policy and trusted application, you can begin testing the AP .NET APIs for protection and unprotection. Before running the sample application, verify the version of the AP .NET installed.

To run the sample application:

> **Note:**
> To run this sample application, ensure that the Application Name in the Trusted Application is set as *Program.exe*.

1. Ensure that the required target framework is configured in the *Program.csproj* file located in `C:\Program Files\Protegrity\Defiance AP\dotnet\sample\Program\` directory.

   For more information about the supported .Net distributions, refer *Supported .Net Distributions*.

2. Run the *GetVersion* API to check the version of the AP .NET installed.
   The following code snippet is used to check the version of the AP .NET.

   ```
   Console.WriteLine("Application Protector version: " + protector.GetVersion());
   ```

3. Navigate to the `C:\Program Files\Protegrity\Defiance AP\dotnet\sample` directory.
4. Open the *Program.sln* file in Visual Studio and run the sample program.
   The following code snippet is from the sample *Program.cs* application.

   > **Important:**
   > Ensure that the data element and the policy user included in the program are a part of the deployed policy.
   >
   > Ensure that the trusted application is added into the data store.

   ```
   using System;
   using System.Collections.Generic;
   using System.Text;
   using Protegrity.Net;
   using Protegrity.PException;

   namespace APDotNetTest
   {
       /
   ********************************************************************************
   ***/
       /**
        * @class   Program
        *
        * @brief   A sample program for Application .NET Protector.
   ```

```
     *
*****************************************************************************************
*******/
    class Program
    {
        private const string dataElementName = "alphanum";
        private const string newDataElementName = "alphanumreprot";
        private const string userName = "policyuser";

        /
*****************************************************************************************
***/
        /**
         * @fn  static void Main(string[] args)
         *
         * @brief   Main entry-point for this application
         *
         * @param   args    An array of command-line argument strings.

*****************************************************************************************
*******/

        static void Main(string[] args)
        {
            Protector protector = null;

            try
            {
                protector = Protector.GetProtector();

                /**
                 * Sample input string data for single operations.
                 */
                string singleInput = "Hello Protegrity";
                byte[] singleByteInput = Encoding.UTF8.GetBytes(singleInput);

                Console.WriteLine("##########################################");
                Console.WriteLine("# Protegrity Application .NET Protector #");
                Console.WriteLine("##########################################\n");

                /**
                 * Calling GetVersion to print APDotNet sdk and Core version.
                 */
                Console.WriteLine(protector.GetVersion() + "\n");
                Console.WriteLine("-------------------------------------");
                Console.WriteLine("-     Single Protect API            -");
                Console.WriteLine("-------------------------------------");
                Console.WriteLine($"Input Data is:        {singleInput}\n");


                /**
                 * Use protector object to call single string Protect API.
                 */
                string protectedData = protector.Protect(singleInput, userName,
dataElementName);
                Console.WriteLine("With String Data Type");
                Console.WriteLine("----------------------");
                Console.WriteLine($"Protected Data is:    {protectedData}");

                /**
                 * Use protector object to call single string Unprotect API
                 */
                string unprotectedData = protector.Unprotect(protectedData, userName,
dataElementName);
                Console.WriteLine($"Unprotected Data is:  {unprotectedData}\n");

                /**
                * Use protector object to call single string Reprotect API.
                */
                string reprotectedData = protector.Reprotect(protectedData, userName,
dataElementName, newDataElementName);
                Console.WriteLine($"Reprotected Data is:  {reprotectedData}");
```

```
                /**
                 * Use protector object to call single string Unprotect API
                 */
                string unprotectReprotectedData = protector.Unprotect(reprotectedData,
userName, newDataElementName);
                Console.WriteLine($"Unprotected Data is:  {unprotectReprotectedData}\n");

                /**
                 * Use protector object to call single byte Protect API.
                 */
                byte[] byteProtectedData = protector.Protect(singleByteInput, userName,
dataElementName);
                Console.WriteLine("With Byte Data Type");
                Console.WriteLine("----------------------");
                Console.WriteLine($"Protected Byte Data is:
{Encoding.UTF8.GetString(byteProtectedData)}");

                /**
                 * Use protector object to call single byte Unprotect API
                 */
                byte[] byteUnprotectedData = protector.Unprotect(byteProtectedData,
userName, dataElementName);
                Console.WriteLine($"Unprotected Byte Data is:
{Encoding.UTF8.GetString(byteUnprotectedData)}\n");

                /**
                 * Use protector object to call single byte Reprotect API.
                 */
                byte[] byteReprotectedData = protector.Reprotect(byteProtectedData,
userName, dataElementName, newDataElementName);
                Console.WriteLine($"Reprotected Byte Data is:
{Encoding.UTF8.GetString(byteReprotectedData)}");

                /**
                 * Use protector object to call single byte Unprotect API
                 */
                byte[] byteUnprotectReprotectedData =
protector.Unprotect(byteReprotectedData, userName, newDataElementName);
                Console.WriteLine($"Unprotected Byte Data is:
{Encoding.UTF8.GetString(byteUnprotectReprotectedData)}");
                Console.WriteLine("\n");

                /**
                 * Sample bulk string input data
                 */
                string[] bulkInput = { "The Alpha-numeric token type tokenizes all
alphabetic symbols (both lowercase and uppercase letters), as well as digits.", "Digits 0
through 9, Lowercase letters a through z, Uppercase letters A through Z", "alphanumeric
data 1234567890 !@#$%^&* with special characters", "ALL THE CHARACTERS IN THIS STRING ARE
UPPERCASE", "UPPERCASE WITH 1234567890 NUMBERS AND !@#$%^&*() SPECIAL CHARACTERS" };
                List<byte[]> byteBulkInput = new List<byte[]>(bulkInput.Length);

                Console.WriteLine("---------------------------------------");
                Console.WriteLine("-         Bulk Protect API           -");
                Console.WriteLine("---------------------------------------");

                Console.WriteLine("Input Data is:");

                /**
                 * Converting string data to byte data.
                 */
                for (int i = 0; i < bulkInput.Length; i++)
                {
                    Console.WriteLine($"{bulkInput[i]}");
                    byteBulkInput.Add(Encoding.UTF8.GetBytes(bulkInput[i]));
                }

                Console.WriteLine("\n");
                Console.WriteLine("With String Data Type");
                Console.WriteLine("----------------------");

                /**
```

```
                 * Use protector object to call bulk string Protect API
                 */
                Tuple<string[], int[]> bulkProtectedData = protector.Protect(bulkInput,
userName, dataElementName);
                Console.WriteLine("Protected Data is: ");
                for (int i = 0; i < bulkProtectedData.Item1.Length; i++)
                {
                    Console.WriteLine(bulkProtectedData.Item1[i] + " " +
bulkProtectedData.Item2[i]);
                }
                Console.WriteLine("\n");

                /**
                 * Use protector object to call bulk string Unprotect API
                 */
                Tuple<string[], int[]> bulkUnprotectedData =
protector.Unprotect(bulkProtectedData.Item1, userName, dataElementName);
                Console.WriteLine("Unprotected Data is: ");
                for (int i = 0; i < bulkUnprotectedData.Item1.Length; i++)
                {
                    Console.WriteLine(bulkUnprotectedData.Item1[i] + " " +
bulkUnprotectedData.Item2[i]);
                }
                Console.WriteLine("\n");

                /**
                 * Use protector object to call bulk string Reprotect API
                 */
                Tuple<string[], int[]> bulkReprotectedData =
protector.Reprotect(bulkProtectedData.Item1, userName, dataElementName,
newDataElementName);
                Console.WriteLine("Reprotected Data is: ");
                for (int i = 0; i < bulkReprotectedData.Item1.Length; i++)
                {
                    Console.WriteLine(bulkReprotectedData.Item1[i] + " " +
bulkReprotectedData.Item2[i]);
                }
                Console.WriteLine("\n");

                /**
                * Use protector object to call bulk string Unprotect API
                */
                Tuple<string[], int[]> bulkUnprotectReprotectedData =
protector.Unprotect(bulkReprotectedData.Item1, userName, newDataElementName);
                Console.WriteLine("Unprotected Data is: ");
                for (int i = 0; i < bulkUnprotectReprotectedData.Item1.Length; i++)
                {
                    Console.WriteLine(bulkUnprotectReprotectedData.Item1[i] + " " +
bulkUnprotectReprotectedData.Item2[i]);
                }
                Console.WriteLine("\n");

                Console.WriteLine("With Byte Data Type");
                Console.WriteLine("----------------------");

                /**
                 * Use protector object to call bulk byte Protect API
                 */
                Tuple<List<byte[]>, int[]> byteBulkProtectedData =
protector.Protect(byteBulkInput, userName, dataElementName);
                Console.WriteLine("Protected Data is: ");
                for (int i = 0; i < byteBulkProtectedData.Item1.Count; i++)
                {

Console.WriteLine(Encoding.UTF8.GetString(byteBulkProtectedData.Item1[i]) + " " +
byteBulkProtectedData.Item2[i]);
                }
                Console.WriteLine("\n");

                /**
                * Use protector object to call bulk byte Unprotect API
                */
                Tuple<List<byte[]>, int[]> byteBulkUnprotectedData =
```

```
protector.Unprotect(byteBulkProtectedData.Item1, userName, dataElementName);
                Console.WriteLine("Unprotected Data is: ");
                for (int i = 0; i < byteBulkUnprotectedData.Item1.Count; i++)
                {

Console.WriteLine(Encoding.UTF8.GetString(byteBulkUnprotectedData.Item1[i]) + " " +
byteBulkUnprotectedData.Item2[i]);
                }
                Console.WriteLine("\n");

                /**
                * Use protector object to call bulk byte Reprotect API
                */
                Tuple<List<byte[]>, int[]> byteBulkReprotectedData =
protector.Reprotect(byteBulkProtectedData.Item1, userName, dataElementName,
newDataElementName);
                Console.WriteLine("Reprotected Data is: ");
                for (int i = 0; i < byteBulkReprotectedData.Item1.Count; i++)
                {

Console.WriteLine(Encoding.UTF8.GetString(byteBulkReprotectedData.Item1[i]) + " " +
byteBulkReprotectedData.Item2[i]);
                }
                Console.WriteLine("\n");

                /**
                * Use protector object to call bulk byte Unprotect API
                */
                Tuple<List<byte[]>, int[]> byteBulkUnprotectReprotectedData =
protector.Unprotect(byteBulkReprotectedData.Item1, userName, newDataElementName);
                Console.WriteLine("Unprotected Data is: ");
                for (int i = 0; i < byteBulkUnprotectReprotectedData.Item1.Count; i++)
                {

Console.WriteLine(Encoding.UTF8.GetString(byteBulkUnprotectReprotectedData.Item1[i]) + " "
+ byteBulkUnprotectReprotectedData.Item2[i]);
                }
                Console.WriteLine("\n");

            }
            catch (ProtectorException e)
            {
                Console.WriteLine(e);
            }
            finally
            {
                if (protector != null)
                {
                    protector.FlushAudits();
                }

            }
        } /* End scope of main function */

    } /* End scope of class */

} /* closure of namespace */
```

Chapter 10

Appendix A: Multi-node Application Protector Architecture

This section describes the multi-node Application Protector architecture, its individual components, and how logs are collected using the Log Forwarder.

The following figure describes the multi-node Application Protector architecture.

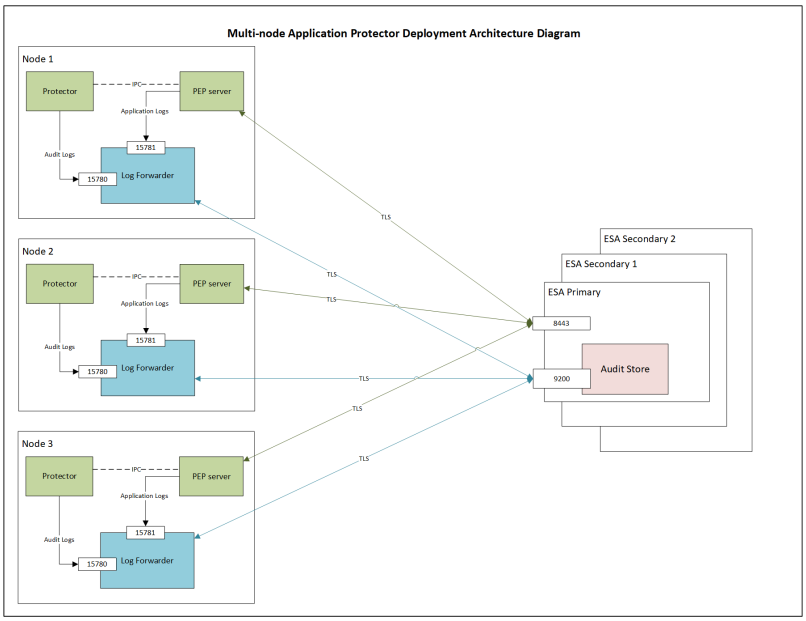

Figure 10-1: Multi-node Application Protector

Consider an example where a few Application Protector (AP) nodes are connected to an ESA, which includes the Audit Store component. Each Application Protector node contains the Log Forwarder and Application Protector instance for sending the logs to the ESA.

> **Note:** For more information about the Log Forwarder, refer to the section *Components of the Logging Architecture* in the *Protegrity Log Forwarding Guide 9.1.0.5*.

The Log Forwarder component collects the logs from the Application Protector and forwards them to the Audit Store. The Log Forwarder uses ports, such as, *15780*, which is configurable and *15781*, which is not configurable, to transport protection and audit logs to the ESA. The ESA receives the logs and stores it in the Audit Store.

For the Application Protector, the configurations of the Log Forwarder can be done using the *pepserver.cfg* file.

> **Note:** For more information about configuring the Log Forwarder in the *pepserver.cfg* file, refer to the section *Appendix: PEP Server Configuration File* in the *Installation Guide 9.1.0.4*.

The ESA communicates with the Log Forwarder using the port *9200*. It is mandatory to configure the Log Forwarder before starting the PEP server to ensure that the Application Protector functions smoothly.

Chapter 11

Appendix B: Using Go Module with Private GitLab Repository

This section describes the steps to use the Go module with a private GitLab Repository.

To set up the Go module in a private GitLab repository:

1. Create a GitLab Personal Access Token with at least *read_api*, *read_repository*, and *write_repository* scopes.

   For more information about creating a personal access token, refer to the section *Creating a personal access token* in the GitLab documentation.

2. Create a *.netrc* file and place it in your home directory.

```
machine privaterepo.example.com
       login user.name@example.com
       password <PERSONAL_ACCESS_TOKEN>
```

3. Run the following Go environment command:

```
go env -w GOPRIVATE=privaterepo.example.com/app/*
```