# Protegrity Application Protector Guide 7.2.1

Created on: Sep 8, 2021

# Notice

## Copyright



Protegrity products are protected by and subject to patent protections;

Patent: *https://support.protegrity.com/patents/*.

Protegrity logo is the trademark of Protegrity Corporation.

NOTICE TO ALL PERSONS RECEIVING THIS DOCUMENT

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

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

Linux is a registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

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

Hadoop or Apache Hadoop, Hadoop elephant logo, Hive, Presto, and Pig are trademarks of Apache Software Foundation.

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

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

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

Pivotal HD is the registered trademark of Pivotal, Inc. in the U.S. and other countries.

# 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 *Protegrity APIs, UDFs, and Commands Reference Guide 7.2.1*.

## 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.

## 1.2 Accessing the Protegrity Documentation Suite

The Protegrity Documentation Suite can be accessed in HTML format from the ESA Web UI.

## To access the Protegrity Documentation Suite:

1. The Protegrity Documentation Center can be accessed from the ESA Web UI by clicking ⓘ > **Help.**
2. Click the ⓘ icon to display the menu.
3. Click **Help** to navigate to Protegrity Documentation Center.
   The following figure shows the Protegrity Documentation Center:

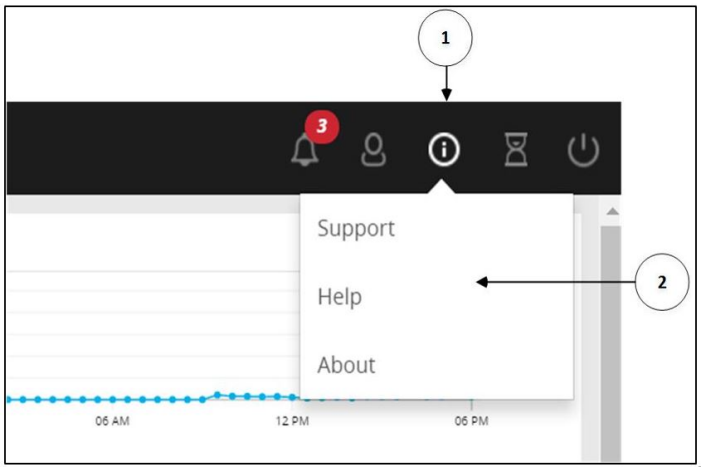


*Figure 1-1: ESA Web UI*

4. Click the icon to display PDF guide.
5. Click the + icon to display Chapter and Related Guide Details.
6. Enter text in **Search** text box to search in all Protegrity guides and display the result.

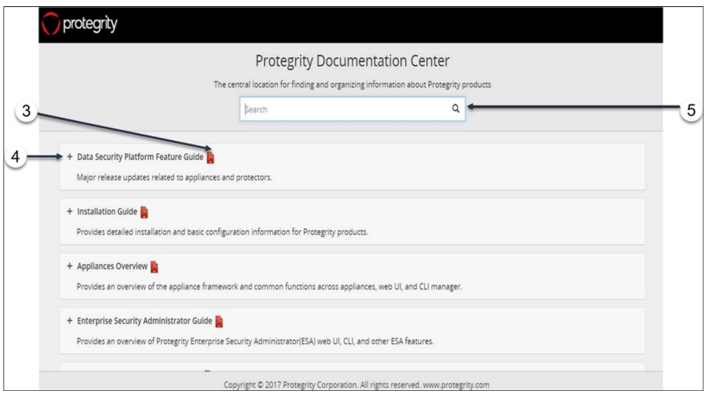


*Figure 1-2: Protegrity Documentation Center*

# Chapter 2

# Protegrity Application Protector (AP) Implementation



## 2.1 Introduction to Protegrity AP

Protegrity AP provides developers with a packaged interface to integrate their applications into the corporate security environment. Protegrity AP delivers all the security policy, cryptographic functions, auditing, reporting and alerting that security environments require.

**Straight Forward Interface**

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

**Separation of Duties**

Security regulations require that there is a clear separation of duties between Security Administrators and application developers. The concept of separation of duties is typically extended to require that 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 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 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.

Protegrity AP 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. Protegrity AP delivers fine grained auditing that can log all access, only encryption/decryption access, tokenization access, or 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. Protegrity Data Security Platform deploys patented key management technology that securely manages all aspects of encryption key operations in the Protegrity AP services.

Because 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 corporate headquarters for use in other applications or stored in databases that are also protected by Protegrity Data Security Platform.

**AP Server Encryption and Tokenization**

The Protegrity AP Server uses the Protegrity Data Security Platform policy and encryption or tokenization settings providing protection for the data elements. AP Server supports the following protection algorithms: 3DES, AES-128, AES-256, CUSP 3DES, CUSP AES-128, CUSP AES-256, DTP2-3DES, DPT2-AES-128, DTP2-AES-256, DTP2-SHA, FPE-FF1, HMAC-SHA1, Token Element, No Encryption.

## 2.2 Protegrity AP API Implementation

The Protegrity AP API has been 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 HMAC hash value. The API is used on supported program languages.

The APIs in AP Client, AP Standard, and AP Java are used to connect and disconnect to the Protegrity PEP Server, whether directly or by using the shared memory as interface, and then to encrypt, decrypt and re-encrypt data. The following additional APIs are used to:

- Check access rights for a user
- Get version of the AP API module used
- Get default key ID for protected data
- Get key ID from protected data
- Retrieve the current key ID used when protecting data

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

**AP Standard** uses the PEP Server for fast data protection with policy check. AP Standard is installed on the same machine as the PEP Server.

**AP Client** uses TCP/SSL connection to the PEP Server for remote encryption/decryption with a policy check.

**AP Lite** is a solution for performing encryption only. The protected key file used for encryption is exported using the REST API tool. Policy checking or tokenizing data is not possible with this AP. It is possible to only encrypt data.

**AP Java** uses the PEP Server with the same version number to provide protection functionality in a Java application. The check access API is not required by this AP as the API can be accessed only by trusted applications.

For more information about the AP APIs, refer to *Protegrity APIs, UDFs, and Commands Reference Guide 7.2.1*.

## 2.3 Limitations

External IV is used, prior to protection, as input to modify the data to protect. The external IV is ignored when using encryption.

# Chapter 3

# Application Protectors C



This section explains the API for Protegrity Application Protectors. AP Standard, AP Client and AP Lite use the same API implemented on C using either Windows or UNIX.

AP Lite is an Application Protector with the functionality limited to protection operations, and therefore its API is restricted. The AP Lite supports only these operations:

- AES encryption
- DES3 encryption
- DTP2 encryption with these options: AES128, AES256, DES3, and SHA1-HMAC.

> **Note:** AP Lite does not generate audits and does not support the following:

- EMPTY handling as the other protectors. If EMPTY data is the input, then the API returns without any processing.
- Protection by tokenization.

## 3.1 AP Deployment Architecture

This paragraph explains the difference between the deployment architecture of the Protegrity Application Protectors.

The following figure explains the AP Client deployment architecture. Remote data encryption/decryption with a policy check is performed over TCP/SSL connection to the PEP Server.

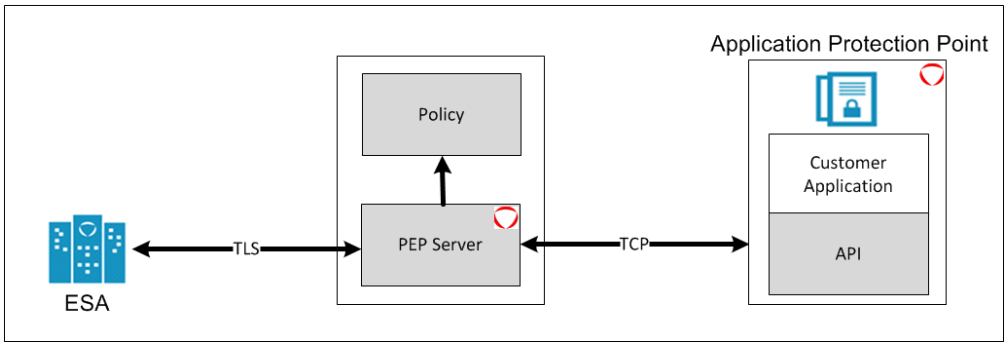


*Figure 3-1: AP Client Deployment Architecture*

The following figure shows the deployment architecture for the AP Standard application protector. AP Standard is installed on the same machine as the PEP Server. The advantage of such deployment is that protection and policy check are performed much faster and there is no latency. The same architecture is applicable to AP Java.

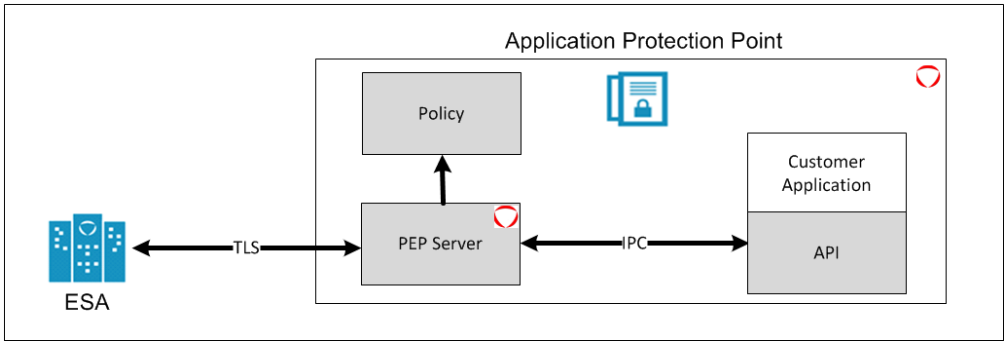


*Figure 3-2: AP Standard/ AP Java Deployment Architecture*

The following figure shows the deployment architecture for the AP Lite application protector. AP Lite uses encryption keys from a BLOB file or an XML file, exported using the REST API tool. Data protection operations are done within the calling process. Only encryption can be performed and no decryption.

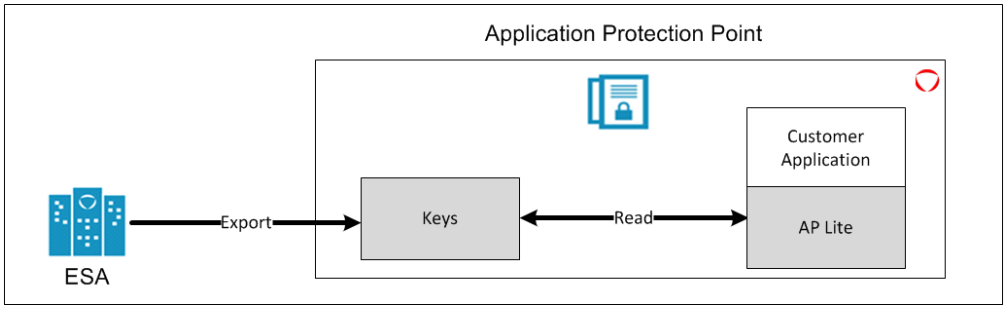


*Figure 3-3: AP Lite Deployment Architecture*

# 3.2 AP Lite Setup

AP Lite only supports protect operations and does not generate audit logs. AP Lite uses data encryption keys from Data Elements defined on ESA, but to use those encryption keys they must first be exported from ESA and saved to a file. AP Lite then uses this *.xml* file to perform the data protection operations.

AP Lite uses keys that are exported as a Binary Large Object (BLOB), in the form of an *.xml* file. The *<export_keys_filename>.xml* file is used to enable protect operation in AP Lite.

AP Lite has some specific and additional steps that should be done to setup and configure this protector.

Follow the scenario explained here to set up and configure AP Lite on ESA.

**Before You begin:**

Ensure that the required encryption data elements are created in ESA. For more information about creating data elements, refer to *Policy Management Guide 7.2.1*. For more information about supported data elements, refer to *Protection Methods Reference Guide 7.2.1*.

Ensure that the user performing the export keys operation is granted the Export Keys permission. You can grant permissions to user or create a specific role with just the required permission. For more information about creating roles or granting permissions, refer *Appliance Overview Guide 7.2.1*.

## 3.2.1 Creating a User with Export Keys Permissions

> **Note:**
> If you need to export data encryption keys from the ESA, then you need to create a user with privileges to export keys from the ESA.
>
> The privileges to Export Keys are granted to the *Security Administrator* role in the ESA, by default.

To create a user with Export Key permissions in the ESA:

1. On the ESA Web UI, go to **Settings** > **Users.**
2. Click the **Roles** tab.
3. Click **Add Role** and create a role with Export Key permissions.

## 3.2.2 Export Keys API

Performing protect operation in AP lite setup requires exporting data encryption keys from the ESA. To perform export keys operation, you must use a client tool that can post HTTPS requests using a REST API. The following section describes the REST API.

**HTTP Method**

POST

**Authentication**

HTTP Basic Auth

**Request URL**

**https://<ESA_IP>:443/dps/v1/management/dataelements/export?format=xml**

> **Note:** The response can either be XML or JSON. The default response format is JSON when format is not defined.

**Request Body**

The request body contains the parameters for exporting the keys, such as the credentials used to protect the exported key values and which data encryption keys to export. The sample request body format can be as follows:

```
{
  "salt": "export",
  "password": "export1234",
  "iterations": 32000
  "dataelements": [
    {
      "name": "DataElement1"
    },
    {
      "name": "DataElement2"
    }
  ]
}
```

In the example, the parameters *salt*, *password*, and *iterations*, which range from 1000 to 32000, assists in deriving a PKCS#5 encryption key that will be used to protect the exported key value. You must define the data element names for which key values need to be exported in the *dataelements* parameter.

**Response**

The following sample response is received when the DataElement1 and DataElement2 are exported.

```
<<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<DATAELEMENTS>
     <DATAELEMENT CIPHERFORMAT="NONE" CURRENTKEYID="2" NAME="DataElement1"><KEY
      ALGORITHM="AES128" CHECKSUM="NONE" IV="NONE" PADDING="PKCS5"
      VALUE="VVlu0MnyV57LwmcbR74v72CGXzVf7iE0Ncqppz2m4yo="/>
      </DATAELEMENT>
      <DATAELEMENT CIPHERFORMAT="NONE" CURRENTKEYID="4" NAME="DataElement1"><KEY
      ALGORITHM="AES256" CHECKSUM="NONE" IV="NONE" PADDING="PKCS5"
        VALUE="vKRpS7Vds+yZW9/lVMa55IQFLy0Rp+wW6K0jc9TlbTJAHwrcs+l3TJQnREZAzlj1"/>
      </DATAELEMENT>
</DATAELEMENTS>
```

The sample response must be saved into a file, namely *<export_keys_filename>.xml* file and must be stored in the same environment where the AP Lite is installed. Ensure that the file is saved in an *.xml* format.

## 3.2.3 Configuring AP Lite on ESA

To set up and configure AP lite:

1. On the target machine, extract the archived installers from the *ApplicationProtector* installation package: *Pepserver* setup file, *XCDevSetup* file, and *XCSamples*.

2. Install XCDevSetup to the default folder. The required libraries (*xcclient.plm, xclite.plm, xcpep.plm*) will be installed in `../protegrity/defiance_xc/bin` folder.
3. Link your application to *xclite.plm* file.
4. Send a post request to export keys in a format as explained in the *Export Keys API* section.
5. Save the response to the *<export_keys_filename>.xml* file.
   You can perform protect operations using the *<export_keys_filename>.xml* file.

# 3.3 BulkProtect Function in C Sharp

A C Sharp sample code, *BulkProtect*, has been provided to call the bulk protect and unprotect APIs provided by the AP-Standard. The sample code is written to perform alphanumeric 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*

## 3.3.1 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:

- Set up Application Protector C on Windows platform. The *BulkProtect* sample code is included in the *\Protegrity\Defiance XC\samples\xccsharpsample* folder. Also, start the PEP Server services for ESA. For more information, refer to the *Installation Guide 7.2.1*.
- Set up the Visual Studio environment and compile the *BulkProtect* sample code into the *BulkProtectSample.exe* file. For more information about compiling the sample code, refer to the *readme.txt* file available within the *xccsharpsample* folder.
- Copy the *xcpep.plm* library from the `\Protegrity\Defiance XC\bin` folder into 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 both protect and unprotect permissions. For more information about deploying policies, refer to the *Enterprise Security Administrator Guide 7.2.1* and *Policy Management Guide 7.2.1*.

## 3.3.2 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. Execute the following command:**`BulkProtectSample.exe`**

The *BulkProtect* sample code first protects the sample data and then unprotects it. The results of both 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 7.1.0.4.master
User        : exampleuser1
Data Element: alphanum


PROTECTING EXAMPLE DATA
================================================
 Input            Output              Error Code
================================================
teststring1234     FgvH5CjnHYkm8Y       6
teststring1234     FgvH5CjnHYkm8Y       6


UNPROTECTING PROTECTED EXAMPLE DATA
================================================
 Input            Output              Error Code
================================================
FgvH5CjnHYkm8Y      teststring1234           8
FgvH5CjnHYkm8Y      teststring1234           8
```

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

## 3.3.3 Walkthrough of the BulkProtect Sample Code

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

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

- *Retrieving the Application Protector version*
- *Initializing libraries*
- *Opening a session*
- *Performing bulk protection*
- *Performing bulk unprotection*
- *Closing the session*
- *Terminating libraries*

### 3.3.3.1 Retrieving the Application Protector Version

This code snippet retrieves the version of the Application Protector and displays 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 );
```

### 3.3.3.2 Initializing Libraries

This code snippet initializes the Application Protector libraries.

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

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

      parameter = communicationId.ToString();
```

### 3.3.3.3 Opening a Session

This code snippet opens a session and returns a handle for that session. This handle is then used in the bulk protect and unprotect APIs. This code snippet 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 );
```

### 3.3.3.4 Performing Bulk Protection

This code snippet is used to bulk protect the data. 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 );
```

### 3.3.3.5 Performing Bulk Unprotection

This code snippet is used to unprotect the data that was protected by the *BulkProtect* API. It displays the input protected data and the output unprotected 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 );
        }
```

### 3.3.3.6 Closing the Session

This code snippet closes the session.

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

### 3.3.3.7 Terminating Libraries

This code snippet terminates the Application Protector libraries.

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

# Chapter 4

# Application Protector Java



Protegrity Application Protector Java (AP Java) is an API that integrates with the client application or user using a common interface to protect data, regardless of where and how the processing is done in the backend.

> **Note:** AP Java Protector has been updated in build 7.0.0.xx and this version of the document applies to AP Java Protector 7.0.0.xx and later.If you are using an earlier version of AP Java Protector, then refer to the document included in the documentation package with build prior to 7.0.0.xx, for API and other related information.

The following figure shows the deployment architecture for the AP Java application protector. AP Java is installed on the same machine where the PEP Server is located. The advantage of the deployment is that protection and policy check are performed much faster with almost no latency. This architecture applies to AP Standard as well.

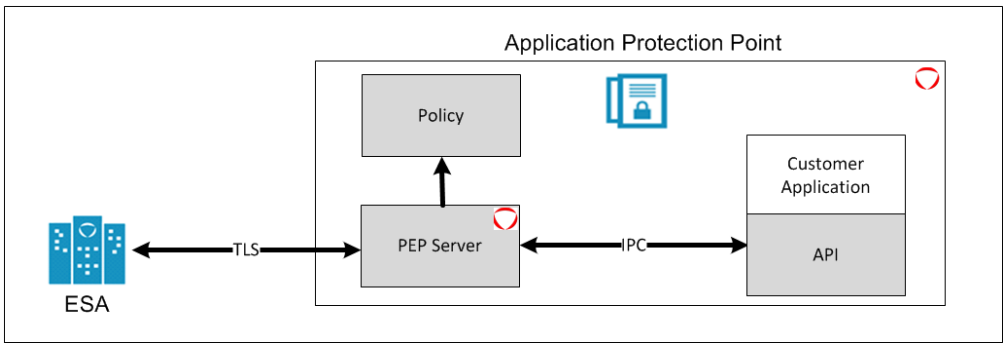


*Figure 4-1: AP Java Deployment Architecture*

AP Java has the following protection and security access methods:

- Get product version
- Get last error
- Get current key id
- Get key id from protected data

- Get default data element
- Protect(including HMAC and bulk operations)
- Unprotect(including bulk operations)
- Reprotect(including bulk operations)

# 4.1 Features

The following are the main features of AP Java.

**Supported Java Distributions**

AP Java supports these Java distributions:

- Java by Oracle Corporation, versions 1.7 and later
- Open JRE, versions 1.7 and later
- IBM J9, versions 1.7 and later

**Trusted Applications**

AP Java can be accessed only by trusted applications. Any application that protects or unprotects data must first be created as a Trusted Application. Refer to *Policy Management Guide 7.2.1*. for detailed steps on how to make an application trusted.

**Session Validity**

A session is only valid until **closeSession** is called or until the period configured in the ApplicationProtectorJava.properties configuration file. The default validity of a session is 15 minutes. An active session is renewed every time the session is used.

**Audit Logs**

AP Java 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 log for every new protection or unprotection method call with a combination of data elements. Logs for 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 Policy Management Web UI. Note that audits are created only after .plm files are loaded. Audits are generated in the ESA Forensics for easy access by the Security Officer.

**Error Handling**

The earlier AP Java provided only Exceptions as error messages. The new AP Java can have three possible types of errors – Success, Failure, and Exception. If there is a Failure error, then using **getLastError** method provides the reason for the failure. For severe errors, Exceptions are raised and the system aborts.

For example, when the AP Java jar file has been configured and loaded, a Boolean is generated:

- Failure = 0, or
- Success = 1

This is an improvement over the earlier AP Java where only Exceptions were raised for all errors and the calling application had to pick up these errors.

Detailed information on all the available methods, how to work with the Application Protector Java API and a How to Get Started example has been provided in the API Javadoc that is included with the application. The API Javadoc is split into three sections.

- *Reference Guide* describes parameters required in the methods and list of the APIs.
- *Programming Guide* walks you through changes that you need to make to the configuration file.
- *Migration Guide* provides steps to migrate from the earlier version of AP Java to this version. Error codes and their definitions are also included.

# 4.2 Improvement over earlier AP Java

This AP Java release has been improved over the earlier Protector.

**Performance**

The performance of AP Java has improved.

**Secure**

The Security Officer must deploy and make applications and users trusted before these users can use AP Java.

All configuration changes and updates are handled only by the Security Officer, who can also view the list of all users that access AP Java API.

Users and applications can configure in which directory the AP Java APIs would be loaded.

**Other Improvements**

Apart from the above, these additional improvements have made AP Java more robust. AP Java supports:

- Bulk data protection and unprotection of up to 1 MB data per call.
- Data types including integer, date, String, float, double, char array and byte array. If the input and output types of the API are byte array, then the customer application should convert the input to and output from the byte array, before calling the API.
- Seeding input data with an external IV (initialization vector) or adding a reference ID before tokenization to get a customized and protected output data.
- Multi-threading and thread-safe operation.
- Improved error handling methods that return one of these – true, false and exception.
- Improved auditing that generates an audit including the data element for each session, for each protect or unprotect or reprotect call.

# 4.3 Link to JavaDoc

All APIs have been documented and explained in the JavaDoc that is included with AP Java. Depending on the platform, JavaDoc can be found in:

*Table 4-1: Location of JavaDoc*

| OS | Location |
|---|---|
| Windows | *C:\Program Files\Protegrity\DefianceAP\java\doc* |
| Unix/Linux | */opt/protegrity/applicationprotector/java/doc* |

# 4.4 Working with AP Java

Code samples for some essential methods are described below.

## 4.4.1 Migrating from existing AP Java

To ensure a faster and more secure AP Java, the PEP Server is now located in the same machine where the application using AP Java is located. The new AP Java uses sessions for protection and unprotection operations. The earlier AP Java did not work this way. For this purpose the existing application source code must be rewritten.

A sample code of the earlier version of AP Java is provided here as an example.

```
ByteBuffer protectByteBuffer    = null;
ByteBuffer unprotectByteBuffer  = "";
String unprotectString          = "";
String inputString              = "abcdefg12345";
XCAPI api                       = new XCAPI();
api.connect( "localhost", 15910, "", "" );
protectByteBuffer = ByteBuffer.wrap( api.encrypt( "policy user", "data element", 8,
                    inputString.getBytes() ) );
unprotectByteBuffer = ByteBuffer.wrap( api.decrypt( "policy user", "data element",
                      8, protectByteBuffer.array() ) );
unprotectString = new String( unprotectByteBuffer.array(), "UTF-8" );
```

For the above sample code, the new AP Java sample would be as follows:

```
String[] inputStringArray      = new String[1];
byte[][] protectByteArray      = new byte[1][];
String[] unprotectStringArray  = new String[1];
inputStringArray[0]            = "abcdefg12345";
Protector api = Protector.getProtector();
SessionObject session = api.createSession( "policy user" );
api.protect( session, "data element", inputStringArray, protectByteArray );
api.unprotect( session, "data element", protectByteArray, unprotectStringArray );
```

## 4.4.2 Deploying a Policy

Java API can be used only by trusted applications.

For more information about deploying a policy, refer to Deploying Data Security Policy in *Policy Management Guide 7.2.1*.

## 4.4.3 Authorizing an Application

For more information about authorizing an application, refer to Working with Trusted Applications in *Policy Management Guide 7.2.1*.

### 4.4.4 Protecting Data

Data can be protected using both encryption and tokenization metho

In the following sample code, the data is a String array "abcdefg12345". After encryption, a byte array of the protected data is returned.

```
String[] inputStringArray      = new String[1];
byte[][] protectByteArray      = new byte[1][];
String[] unprotectStringArray  = new String[1];
inputStringArray[0]            = "abcdefg12345";
Protector api = Protector.getProtector();
SessionObject session = api.createSession( "policy user" );
api.protect( session, "data element", inputStringArray, protectByteArray );
```

### 4.4.5 Unprotecting Data

In the following sample code, the string array “abcdefg12345” is first protected. The protected string is passed as input to the **unprotect** method and clear text is returned. As a prerequisite it must be ensured that the application, policy user, using this method is a trusted application.

```
String[] inputStringArray      = new String[1];
byte[][] protectByteArray      = new byte[1][];
String[] unprotectStringArray  = new String[1];
inputStringArray[0]            = "abcdefg12345";
Protector api = Protector.getProtector();
SessionObject session = api.createSession( "policy user" );
api.protect( session, "data element", inputStringArray, protectByteArray );
api.unprotect( session, "data element", protectByteArray, unprotectStringArray );
```

### 4.4.6 Reprotecting Data

In the following sample code, the string array “abcdefg12345” is first protected using **oldexternalIv**. The protected string is passed as input to the **reprotect** method with **newexternalIv** to return a byte array. As a prerequisite it must be ensured that the application, policy user, using this method is a trusted application.

```
String[] inputStringArray      = new String[1];
byte[][] protectByteArray      = new byte[1][];
byte[][] reprotectByteArray    = new byte[1][];
inputStringArray[0]            = "abcdefg12345";
static byte[] oldexternalIv = "1234".getBytes();
static byte[] newexternalIv = "2635".getBytes();
Protector api = Protector.getProtector();
SessionObject session = api.createSession( "policy user" );
api.protect( session, "old data element", inputStringArray, protectByteArray, oldexternalIv );
api.reprotect( session, "new data element", “old data element”, protectByteArray,
reprotectByteArray, newexternalIv, oldexternalIv );
```

## 4.5 Running AP Java - Example

This section provides a simple example on how to use the AP Java protection and unprotection methods. It is assumed that ESA is already available, and AP Java has been successfully installed.

The tasks can be divided in this order:

- Install AP Java.
- Create data elements and data store in Policy Management Web UI.
- Create users and configure policy.
- Configure the trusted application.
- Use the protection and unprotection APIs by first protecting the string array and then unprotecting the protected value.

## 4.5.1 Installing AP Java

To install AP Java:

AP Java must be successfully installed to protect or unprotect sensitive information. To verify that the installed software is correct, run the **`GetVersion`** method to check the version of the installed AP Java.

**public java.lang.String getVersion()**

The code snippet to check the version number follows.

```
/* 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;
        }

   }
}
```

## 4.5.2 Creating Data Element and Data Store in Policy Management

**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 data elements and data store:

1. Go to the ESA Web UI, then to **Policy Management** > **Data Elements & Masks** > **Data Elements.**
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 *Policy Management Guide 7.2.1*.

## 4.5.3 Configuring Policy

To Configure Policy:

Create a data security policy user and configure the policy.
For more information about creating a data security policy, refer to *Policy Management Guide 7.2.1.*

## 4.5.4 Configuring a Trusted Application

**Before you begin**
To ensure that only applications and users known to ESA are able to access AP Java APIs, these have to be configured as trusted applications under the ESA security policy. If a policy is deployed but the application or a user is not trusted, then while performing any protect or unprotect operation, an error message - `API consumer is not part of the trusted applications, please contact the Security Officer` - is displayed and AP Java aborts.

To configure a new trusted application:

1. On the ESA Web UI, go to **Policy Management** > **Policies & Trusted Applications** > **Trusted Application.**
2. After the trusted application is configured for the application user, add the AP JAVA data store to the trusted application and then save it.
3. Deploy the policy using the Policy Management Web UI.
   For more information about Trusted Applications, refer to *Policy Management Guide 7.2.1*.

## 4.5.5 Linking a Trusted Application to a Data Store

To link a Trusted Application to a data store:

1. On the ESA Web UI, go to **Policy Management** > **Policies & Trusted Applications** > **Trusted Application.**
2. After the trusted application is configured for the application user, add the AP JAVA data store to the trusted application and then save it.
3. Deploy the policy using the Policy Management Web UI.
   For more information about linking a Trusted Application to a data store, refer to *Policy Management Guide 7.2.1*.

## 4.5.6 Deploying the Data Store to the Protector

To deploy the data store to the Protector:

1. On the ESA Web UI, go to **Policy Management** > **Policies & Trusted Applications** > **Trusted Application.**
2. After the AP JAVA data store is added to the trusted application, deploy the data store to the protector using the Policy Management Web UI.
3. Deploy the policy using the Policy Management Web UI.

   For more information about deploying a data store to the Protector, refer to *Policy Management Guide 7.2.1*.

## 4.5.7 Using The AP Java APIs

After you have set up the the policy and trusted application, you can begin testing AP Java APIs for protection, unprotection, and reprotection. A sample follows. Comments are available at every stage to help you.

```
/* 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-1.7-64_7.2.1.x.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) {
       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) {
              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
       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();
           }
       }
}
```

# Chapter 5

# Application Protector (AP) Python



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

## 5.1 Features

The following are the main features of AP Python:

- **Supported Python Distributions**

  AP Python supports the following Python versions:

  - Python 3.7.x for Red Hat Enterprise Linux (RHEL) operating systems

- **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 *Policy Management Guide 7.2.1*.

- **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. This means that in case of single data item calls, three audit log events will be generated if you do one protect operation with data element name *"a"*, five protect operations with data element name *"b"*, and 1000 unprotect operations with data element *"a"*. In case of Bulk data items, every data protection operation, such as, protect, unprotect, and reprotect will generate audit log events. You can use this knowledge to ensure how you want the audit records to be generated, in case of single 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 *Policy Management Guide 7.2.1*.

- **Error Handling**

  The AP Python APIs return an error message in case of policy and system exceptions. If AP Python is used to perform a security operation on a single data item, then an exception appears in case of any error. However, if AP Python is used to perform a security operation on bulk data, then no exception is thrown in case of any error. Instead, an error list is returned for the individual items in the bulk data.

  For more information about the AP Python API error return codes, refer to *Application Protector (AP) Python API Return Codes* section in the *Protegrity Troubleshooting Guide 7.2.1*.

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

  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 *Using AP Python in a Development Environment* section in the *APIs, UDFs, and Commands Reference Guide 7.2.1*.

# 5.2 AP Python Deployment Architecture

The following figure shows the deployment architecture for AP Python. AP Python is installed on the same machine where the PEP server is located. The advantage of this deployment is that the protection and policy check are performed much faster with almost no latency.

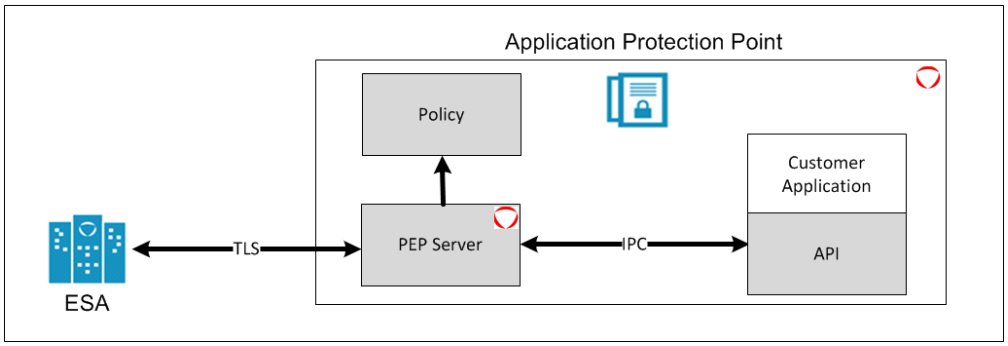


*Figure 5-1: AP Python Deployment Architecture*

## 5.3 AP Python APIs

AP Python has the following protection and security access APIs:

- Get product version
- Get current key id
- Extract key id from data
- Get default data element
- Protect
- Unprotect
- Reprotect
- check_access

For more information about the AP Python APIs, refer to the *APIs, UDFx, and Commands Reference Guide*.

## 5.4 AP Python APIs and Supported Protection Methods

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

> **Note:**
> Starting from the Version 7.1, Maintenance Release 1 (MR1), the DTP2 protection method is deprecated.
>
> For assistance in switching to a different protection method, contact Protegrity.

*Table 5-1: AP Python APIs and Supported Protection Methods*

| Operation | Input | Output | Protection Method Supported |
|---|---|---|---|
| Protect | String | String | Tokenization, No Encryption, FPE |

| Operation | Input | Output | Protection Method Supported |
|---|---|---|---|
| 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, DTP2, CUSP, Printable, Binary, DTP2-Printable, 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, DTP2, CUSP, Printable, Binary, DTP2-Printable, 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, DTP2, CUSP, Printable, Binary, DTP2-Printable, Hashing<br>**Note:**<br>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 not supported for AP Python. A protect, unprotect or a reprotect operation performed by using the Integer token type with 2 bytes as a data element for input data, 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.

[*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.

# 5.5 Running AP Python - Example

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

The tasks can be divided in this order:

- Install AP Python.
- Create data elements and data store in ESA Web UI.
- Create users and configure policy.
- Configure the trusted application.
- Use the *protect*, *reprotect*, and *unprotect* AP Python APIs to protect, reprotect, and unprotect the string value respectively.

## 5.5.1 Installing AP Python

➤ To install AP Python:

AP Python must be successfully installed to protect or unprotect sensitive information. The AP Python installation package is installed using the the *pip* installer available in Python. To verify that the installed software has been correctly installed, run the **`GetVersion`** method to check the version of the installed AP Python.

The following snippet contains the 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())
```

## 5.5.2 Creating Data Elements and Data Store in Policy Management

**Before you begin**
Before you run the application, you must decide whether you want to tokenize or encrypt the data. Accordingly, you can create a Tokenization or Encryption data element.

➤ To create data elements and data store:

1. Navigate to the ESA Web UI, then to **Policy Management** > **Data Elements & Masks** > **Data Elements.**

2. Similarly, to create a data store, navigate to **Policy Management** > **Data Stores.**
   For more information about creating data elements and data stores, refer to the *Policy Management Guide 7.2.1*.

## 5.5.3 Configuring a Policy

➤ To Configure Policy:

1. Navigate to the ESA Web UI.
2. Create a data security policy user and configure the policy.
   For more information about creating a data security policy user and configuring a policy, refer to *Policy Management Guide 7.2.1.*

## 5.5.4 Configuring a Trusted Application

**Before you begin**
To ensure that only applications and users known to ESA are able to access the AP Python APIs, these have to be configured as Trusted Applications under the ESA security policy. If a policy is deployed but the application or a user is not trusted, then while performing any protect or unprotect operation, the error message `API consumer is not part of the trusted applications, please contact the Security Officer` appears and AP Python aborts.

➤ To configure a new Trusted Application:

1. On the ESA Web UI, navigate to **Policy Management** > **Policies & Trusted Applications** > **Trusted Application.**
2. After the Trusted Application is configured for the application user, add the AP Python data store to the trusted application and then save it.
3. Deploy the policy.
   For more information about Trusted Applications and deploying a policy, refer to *Policy Management Guide 7.2.1*.

## 5.5.5 Linking a Trusted Application to a Data Store

➤ To link a Trusted Application to a data store:

1. On the ESA Web UI, navigate to **Policy Management** > **Policies & Trusted Applications** > **Trusted Application.**
2. After the trusted application is configured for the application user, add the AP Python data store to the trusted application and then save it.
3. Deploy the policy.
   For more information about linking a Trusted Application to a data store and deploying a policy, refer to *Policy Management Guide 7.2.1*.

## 5.5.6 Deploying the Data Store to the Protector

To deploy the data store to the Protector:

1. On the ESA Web UI, navigate to **Policy Management** > **Policies & Trusted Applications** > **Trusted Application.**
2. After the AP Python data store is added to the trusted application, deploy the data store to the protector.
3. Deploy the policy.

   For more information about deploying a data store to the Protector and deploying a policy, refer to *Policy Management Guide 7.2.1*.

## 5.5.7 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.

```
# -*- 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)
```