opentext™

OpenText™ Information Archive

Encryption Guide

Configure how data is encrypted, according to your organization's needs and security considerations. Learn about encryption, decryption, key generation, and management.

EARCORE250400-AGE-EN-01

OpenText™ Information Archive
Encryption Guide

EARCORE250400-AGE-EN-01
Rev.: 2025-Sept-08

This documentation has been created for OpenText™ Information Archive CE 25.4.
It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product, on an OpenText website, or by any other means.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111
Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440
Fax: +1-519-888-0677
Support: https://support.opentext.com
For more information, visit https://www.opentext.com

Table of Contents

Chapter 1

Overview

The OpenText Information Archive cryptography module provides cryptographic capabilities (encryption, decryption, key generation and management) for data at rest, including sensitive application data, passwords, credentials, search results, and so on. You can configure how data is encrypted, according to your organization's needs and security considerations.

These cryptographic capabilities leverage state-of-the-art security providers (SunJCE, Bouncy Castle, and Gemalto (Thales CipherTrust Manager, formerly SafeNet KeySecure), which are accessed by a standard *Sun Java Cryptography Extension* (JCE) *Java Cryptography Architecture* (JCA) architecture.

Notes

- Depending on your country, encryption might not be available due to regulations. The content in this guide is not applicable to the OpenText Information Archive NO ENCRYPTION product.
- When the generic term Gemalto appliance is used in this guide, it is referring to either Thales CipherTrust or Safenet KeySecure. As of the publication of this guide, both products are owned by Gemalto.

The cryptography module allows users to:

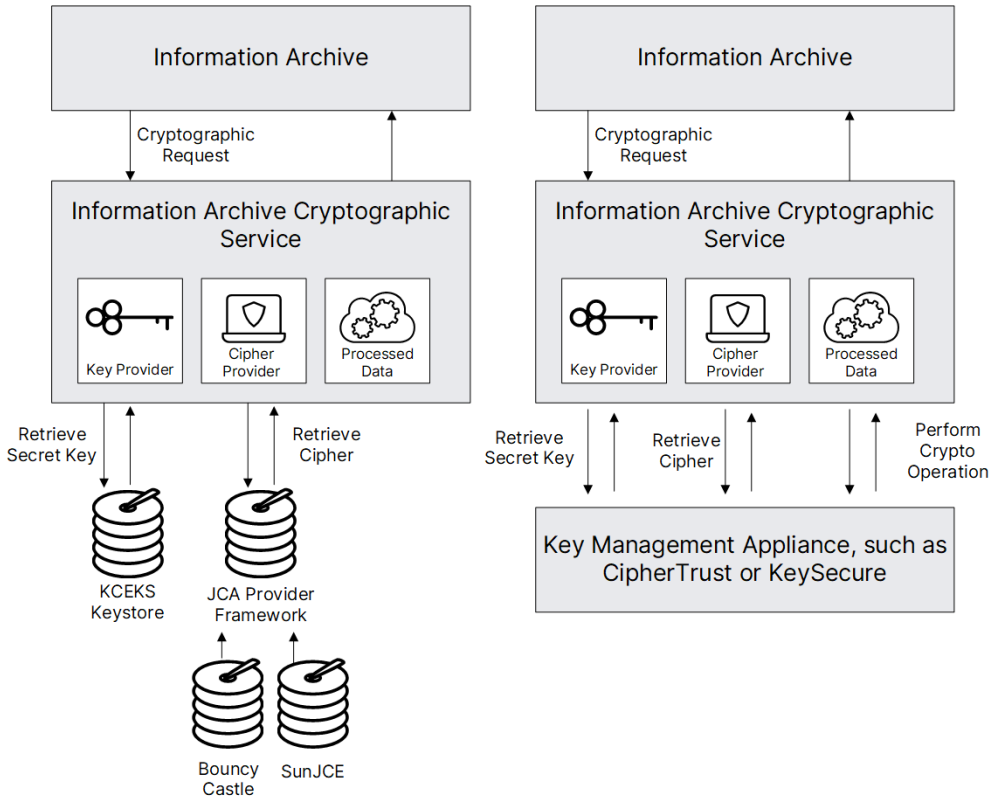
- Configure encryption for passwords and keys to be used in OpenText Information Archive modules (for example, the `application.yml` configuration files used by IA Server, IA Web App, IA Shell, and so on).
- For SIP applications, apply encryption to data and content during ingestion. Encryption can be applied to:
  - Structured data contained in the `eas_pdi.xml` file
  - Unstructured contents contained in a SIP package
  - A SIP package once it has been received by OpenText Information Archive
  - The contents of an AIP
- For table applications, apply encryption to structured data and unstructured content during ingestion, and decrypt data during a search.

For example, personally identifiable information (PII) can be encrypted during ingestion. The search screen allows valid users to search encrypted data. Results are displayed in an encrypted or decrypted format for the configured groups.

The following diagram illustrates how OpenText Information Archive handles PII encryption for SunJCE or Thales CipherTrust Manager/Gemalto SafeNet KeySecure.

OpenText Information Archive makes cryptographic requests to the OpenText Information Archive Cryptographic Service, which provides the following cryptographic functionality to other modules:

- Encryption and decryption of byte arrays
- Encryption and decryption of input streams
- Hashing of values against a salt
- Verification of a hash value against a provided value



OpenText Information Archive supports the following two types of key storage capabilities:

Java Keystore	Gemalto SafeNet KeySecure
• The path to the keystore is specified in OpenText Information Archive's configuration files. • You must make sure that the keystore is adequately protected. • You can store the crypto keystore in a database.	• The path to the properties file for Thales CipherTrust/KeySecure appliance client's is specified in OpenText Information Archive's configuration files. The system's components act as appliance's clients so the properties file does not need to be in a shared location. • You manage the properties file for CipherTrust/KeySecure's client.

1.1 Performance for data encryption and decryption

Data encryption and decryption are additional steps in ingestion and search that take CPU time and use memory. If you use encryption in the archive process, then you should pay attention to performance.

If encryption is not required for an application, then do not include it in the application's configuration.

Sufficient entropy must be available on the host running each OpenText Information Archive component, otherwise the system might slow substantially, as it must wait for sufficient entropy to become available again. This issue can occur on any system but is more immediate on virtual machines. For Linux users, to avoid pauses in encryption configuration during ingestion, one possible solution is to run the Haveged daemon or the `rng-tools` package on the Linux server. Using a hardware-based way of introducing sufficient entropy or any other secure means of supplying additional entropy will also work well. Alternatively, with JDK 11 Java processes on Unix-like operating systems can be started by specifying additional JVM option `java.security.egd` to point to `/dev/./urandom` to use a non-blocking entropy source:

```
-Djava.security.egd=file:/dev/./urandom
```

1.2 Data encryption limitations

- Only Equal or Not Equal search operators can be used on encrypted data.
- Data must be decrypted with the same settings that were used to encrypt it. If you change the encryption settings after ingesting data, the ingested data must be decrypted with the original encryption settings.
- For more information about limitations for SIP ingestion, see PDI crypto.

1.3 Cryptography providers, algorithms, modes, and padding for various ciphers

Cryptography is a complex subject which involves algorithms, ciphers, providers, modes, paddings, and so on. There are many different standards, each providing different capabilities and strengths.

OpenText Information Archive currently supports one algorithm, AES, and three security providers: SunJCE, Bouncy Castle and Gemalto. Among these implementations there is some flexibility with respect to the mode and padding of the actual transformation and encryption process. Not all of the combinations work, so the following compatibility matrix clarifies the combinations that do work.

Security Provider	Modes	Supported Paddings
SunJCE	• CBC • PCBC • CFB • OFB	• ISO10126Padding • PKCS5Padding • NoPadding
	• CTR • CTS	• PKCS5Padding • NoPadding
Bouncy Castle	• CBC • CTS • OpenPGPCFB	• ISO10126Padding • PKCS5Padding • NoPadding • X9.23Padding • ISO7816-4Padding • PKCS7Padding • WithCTS
	• OFB • CFB • SIC • GCFB	• ISO10126Padding • PKCS5Padding • NoPadding • X9.23Padding • ISO7816-4Padding • PKCS7Padding
	• EAX • GCM	• NoPadding
Gemalto	• CBC	• PKCS5Padding • NoPadding

1.4 Required configuration after installing the product and encrypting passwords

While the OpenText Information Archive installation process automatically generates several configuration files, some configuration files still require you to manually enter the passwords.

Whether these values need to be populated will depend on how the system is configured. When these value are, in fact, required, it is recommended that you update the following configuration files with the appropriate plain-text password values *before* encrypting the passwords using setup. This way, all the passwords are encrypted properly.

IA Server

Assuming you are using OTDS, the setup program does not automatically set the OTDS password. Modify the file and set the value; otherwise, it will not be possible to update the group role mappings from the IA Web App (under **Administration > Groups**).

Be sure to enter the `password` in the `iaserver/application-otds.yml` file:

```
OTDS:
  LOCATION:
    authUrl: ${OTDS.LOCATION.protocol}://${OTDS.LOCATION.host}:$
{OTDS.LOCATION.port}/otdsws/oauth2/token
    host:
    httpsPort:
    path: /otdsws/rest
    port:
    protocol: http
    url: ${OTDS.LOCATION.protocol}://${OTDS.LOCATION.host}:${OTDS.LOCATION.port}$
{OTDS.LOCATION.path}
  infoarchive:
    clients:
      server:
        clientId: infoarchive.server
        clientSecret:
    resource:
      id: infoarchive
  password:
  username:
security:
  oauth2:
    resource:
      jwt:
        keyValue: NOTUSED
```

The `clientSecret` is only used for Content Aviator support, which is only supported for Cloud deployments.

IA Web App

Assuming you are using OTDS, enter the information for the highlighted property in the `iawebapp/application-infoarchive.gateway.profile.OTDS.yml` file. If you do not specify the `password` for OTDS, the IA Web App startup will fail:

```
OTDS:
  infoarchive:
    clients:
```

```
      cli:
        timeout: 200000
      gateway:
        clientId: infoarchive.gateway
        clientSecret:
        scope: otds:groups
      iawa:
        clientId: infoarchive.iawa
        clientSecret:
        logoutUrl: ${OTDS.infoarchive.gateway.protocol}://$
{infoarchive.gateway.host}:${infoarchive.gateway.port}$
{infoarchive.gateway.contextPath}logout
        scope: otds:groups
    gateway:
      logoutSuccessUrl:
      protocol: http
    resource:
      id: infoarchive
  location:
    contextPath: /otdsws
    host: localhost
    httpsHost: ${OTDS.location.host}
    httpsPort: 8443
    logoutUrl: https://${OTDS.location.httpsHost}:${OTDS.location.httpsPort}$
{OTDS.location.contextPath}/logout
    path: ${OTDS.location.contextPath}/rest
    port: 8090
    url: http://${OTDS.location.host}:${OTDS.location.port}${OTDS.location.path}
  password:
  urlEncodeBeforeBase64: true
  username: otadmin@otds.admin
```

The `clientSecret` values for Gateway and the IA Web App are prompted by setup.

IA Shell

If you plan to store the credentials for connecting, specify the password before encrypting.

If you do not specify the username or password, you will be prompted for credentials when using IA Shell or installing applications (either example applications or first time applications).

To have your credentials saved, enter the highlighted properties in the `iashell/application.yml` file:

```
connection:
  clientSecret:  <your client secret>
  gatewayUrl: http://10.9.203.252:8080
  tenant: INFOARCHIVE
 userName: <YOUR USER NAME>
  userPassword: <YOUR PASSWORD>
```

Important

It is recommended to not specify the `userName` or `userPassword` in the IA Shell's `application.yml` file unless you are the only user who accesses that machine.

Chapter 2

Managing cryptographic keystores

OpenText Information Archive components can use several different types of keystores to protect data with cryptography:

Type of Data Protected	Keystore Types Supported	Keystore Locations Supported	Notes
Sensitive data in the system data database and search results, using the **default crypto object**	• JCEKS • PKCS11 • PKCS12 • BKS	• Database	The default crypto object uses the crypto keystore, which is stored in the database.
Data at rest, using a **crypto object**	• JCEKS • PKCS11 • PKCS12 • BKS	• Database • Thales CipherTrust / Gemalto SafeNet KeySecure	Each crypto object specifies which security provider to use. If the security provider is `Gemalto`, then the symmetric key associated with the crypto object is managed by CipherTrust/ KeySecure appliance. Otherwise, the symmetric key is managed by the crypto keystore, which is stored in the system database.
Password encryption for component passwords and secret tokens in configuration files	• JCEKS • PKCS11 • PKCS12 • BKS	• Local filesystem • Network filesystem • CipherTrust / KeySecure appliance	It is strongly recommended that you use a separate keystore for password encryption, rather than storing the keys for password encryption in the crypto keystore.
Data in transit, using **TLS/SSL**	• JKS • JCEKS • PKCS11 • PKCS12	• Local filesystem • Network filesystem	It is strongly recommended that for TLS/SSL, any keystores are separate files from any truststores.

2.1 Secret keys and keystores

There is always one keystore that stores the secret key used for cryptography related to sensitive data attributes of SDPG objects, as well as secret keys used for search results (one per application). This is known as the crypto keystore, and it always resides in the system database.

Additional secret keys, used for cryptography related to content, data, and so on, can be stored in that same keystore, or they can be stored in CipherTrust/KeySecure appliance. Where these secret keys are stored depends on the configuration details of their crypto object. For more information about crypto objects, see Cryptography objects.

2.2 Accessing the keystores

Each instance of the same component must be able to access that component's keystore. For example, multiple instances of IA Server must be able to access the IA Server keystore. If this access is not enabled, then encrypted data in one instance cannot be decrypted by a different instance. For this reason, in a multi-server environment, you must not store the keystores on a local filesystem.

2.3 Keystores and configuration files

You can specify the location of the keystores, as well as the passwords that protect them, using parameters in the configuration files. You should use a strong, randomly generated password to protect each keystore, and you should encrypt the passwords that are stored in configuration files. You should also make sure that the keystores can only be accessed by authorized users, and the OpenText Information Archive components are running as users that can read the files.

2.4 Specifying the location of the keystores

You specify the location of the keystores when you install OpenText Information Archive. You can choose to enable password encryption, data encryption, and the encryption of data in transit using TLS/SSL.

2.5 The keystore for encrypted passwords

It is required that the keystore for encrypted passwords be in a dedicated keystore. If the crypto keystore is stored in a database, it cannot be used to store the keys for encrypting and decrypting passwords.

2.6 The keystore for crypto objects and the default crypto object

You can also use CipherTrust/KeySecure appliance as a keystore for a crypto object, but not for the default crypto object. If you do this, you effectively have two keystores: the crypto keystore and CipherTrust/KeySecure store.

When the crypto keystore is stored in a database, you cannot also store keys used for encrypting component passwords and secret tokens in the same keystore. You must use a different keystore for the passwords and tokens.

Caution

- You should apply additional protections to the crypto keystore. If the crypto keystore is stored in a database, you should set up TLS/SSL to protect data in transit and make sure that only trusted clients are allowed to connect to the database. If the crypto keystore is stored on a local or network filesystem, you should restrict access to the file using filesystem permissions.
- You must store the crypto keystore in a database or use CipherTrust/KeySecure appliance with crypto objects.

If you want to make Gemalto a security provider option for crypto objects, you must connect OpenText Information Archive components to CipherTrust/KeySecure appliance and configure the crypto objects accordingly. For more information, see Connecting system components to CipherTrust/KeySecure appliance and Managing crypto objects.

2.7 Managing secret keys

For sensitive data and data at rest, OpenText Information Archive uses the default crypto object and other crypto objects to create secret keys. These crypto objects track which key to use for data by specifying the key ID, algorithm, key size, and so on. The keys are actively managed by OpenText Information Archive, and you should not change them manually.

You can use a key allotment scheme for data ingestion so that OpenText Information Archive switches secret keys after a period of time (for example, after a week or six months). When new data is ingested, OpenText Information Archive uses the latest keys, without changing any of the keys for data that has already been ingested. For more information, see Allotting encryption keys.

For data in transit, TLS/SSL stores its public key certificates and corresponding private keys on a local or network filesystem. These keys are not used to encrypt data at rest. The system does not manage these keys directly, but relies on the keys being in the keystores as configured. Apart from this, you can manage the keys however you want.

For component passwords and secret tokens that are persistently stored in configuration files, you can specify whether a component's passwords are encrypted, as well as other parameters. Each component manages this on an individual basis and could use either the same or different keys. The password encryption tool uses its cryptography parameters when encrypting passwords, and OpenText Information Archive components use their cryptography parameters when decrypting the passwords.

Note: When decrypting passwords, you will be prompted to enter the gateway alias and keystore password.

As with data in transit, OpenText Information Archive does not manage these keys directly, but relies on the keys being in the keystores as configured. Apart from this, you can manage the keys however you want, given that the keys stay the way they are, with no editing, deleting, or replacing. If a key is modified in any way, the system might not be able to use the key to decrypt data that was encrypted by the original key.

Secret keys are also stored in a dedicated password encryption keystore, which is password protected. This password must be encrypted, but to encrypt and decrypt the password, the key must either be provided during the startup of the individual components or persistently stored on disk to support silent startup. For more information, see Starting IA Server, IA Web App, and IA Shell without password input.

During the installation of OpenText Information Archive components, the setup application processes references to keystores and passwords only. The install software does not manage SSL or general crypto-related keystores, but it does manage the password encryption keystore and it does encrypt and decrypt passwords across the configuration files.

Chapter 3

Connecting system components to CipherTrust/ KeySecure appliance

OpenText Information Archive components can use Bouncy Castle, SunJCE, or Gemalto as a security provider. However, to use Gemalto with these components, there are additional steps that you must perform.

Gemalto - Thales CipherTrust or SafeNet KeySecure are key management platforms that can also encrypt and decrypt data. OpenText Information Archive uses Gemalto as a key generation and management utility only.

OpenText Information Archive components can use Gemalto as follows:

- IA Server: for data encryption and decryption, as well as password decryption
- IA Web App and IA Shell: for password decryption
- The password encryption utility: for password encryption

There are two scenarios where you can connect OpenText Information Archive components to Gemalto:

You are performing a new installation of OpenText Information Archive
You can connect an OpenText Information Archive component to Gemalto before or after you install the OpenText Information Archive component. If you are performing a new installation of OpenText Information Archive, it is easier to connect an OpenText Information Archive component to Gemalto before you install the OpenText Information Archive component.

You have an existing installation of OpenText Information Archive
Follow the steps below to connect an OpenText Information Archive component to Gemalto after you install the OpenText Information Archive component.

The Gemalto client libraries (also called Ingrian) are not bundled with OpenText Information Archive. You must obtain the license and libraries from the vendor, Gemalto, a Thales company. OpenText Information Archive provides a mechanism so that IA Server, IA Web App, IA Shell, and the password encryption utility can load external libraries.

Depending on how you set up IA Web App (as either a standalone Spring Boot application or deployed to Apache Tomcat), you must perform different steps to connect IA Web App to Gemalto appliance.

Before you begin:

- Obtain a license for Thales CipherTrust or Gemalto SafeNet KeySecure from Gemalto, a Thales company.

- Obtain the SafeNet KeySecure ProtectApp JCE (Java) software bundle from Gemalto, a Thales company.
- Make sure that the software bundle includes `IngrianNAE-8.12.2.000.jar` or a later version of the file.
- Obtain important CipherTrust/KeySecure documents:
  - The *ProtectApp JCE User Guide* / *CipherTrust Application Data Protection JCE User Guide*, which includes information about additional configuration parameters in the `properties` file.
  - The *KeySecure Admin User Guide* / *CipherTrust Manager Administration*, which includes information about configuring users and groups, certificates, and the key server (NAE-XML).
- Make sure the CipherTrust/KeySecure "Key Server" configuration for NAE-XML is in place and includes the following:
  - Allow Key and Policy Configuration Operations – must be enabled
  - Allow Key Export – must be enabled
- On Thales CipherTrust or Gemalto SafeNet KeySecure, do the following:
  1. Create a local user.
  2. Create a local group and add the user from step 1.

Note: You must perform the steps below on each computer that hosts an OpenText Information Archive component.

To configure Gemalto for an OpenText Information Archive component:

This procedure can be done before or after running the setup program.

1. Copy the IngrianNAE JAR file from the CipherTrust/KeySecure software bundle to the `<IA_ROOT>/lib/iasetup` directory.

   Note: If you already ran the setup program, instead copy into the required directory, depending on which component you are installing:

   - For IA Server: `<IA_ROOT>/lib/iaserver/external`
   - For IA Web App installed as a standalone Spring Boot application: `<IA_ROOT>/lib/iawebapp/external`
   - For IA Web App deployed to Apache Tomcat: `<TOMCAT_ROOT>/webapps/infoarchive-webapp/WEB-INF/lib`
   - For IA Shell: `<IA_ROOT>/lib/iashell/external`

2. Create a sub-directory for Gemalto in the shared resources location (for example, `\\server01\iadata\gemalto`). For more information about the shared resources location, see section 1.2 "System directories" in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*.

3. Copy the `IngrianNAE.properties` file from the CipherTrust/KeySecure software bundle to the sub-directory that you just created in the shared resources location (`<SHARED_RESOURCES_DIR>/gemalto`).

4. In a text editor, open the `<IA_ROOT>/shared/IngrianNAE.properties` file and configure it according to how CipherTrust/KeySecure has been deployed in your environment. At a minimum, you must specify the following parameters:

   `NAE_IP`
   The IP address of the CipherTrust/KeySecure server.

   `NAE_Port`
   The port that CipherTrust/KeySecure server runs on.

   `Protocol`
   The protocol used to connect to the CipherTrust/KeySecure server, either `tcp` or `ssl`. SSL is recommended because it is more secure.

   `Key_Store_Location`
   The absolute path of the truststore. At a minimum, the truststore should contain the CipherTrust/KeySecure server's SSL certificate to establish trust for the client, and the client's SSL certificate that will be used by OpenText Information Archive to establish an SSL connection.

   `Symmetric_Key_Cache_Enabled`
   Enables a cache for fetching symmetric keys from CipherTrust/KeySecure appliance, which prevents a performance bottleneck that can slow data ingestion significantly.

   You should set this parameter to `yes`.

   `Symmetric_Key_Cache_Expiry`
   The number of seconds before the cache expires. You should set this parameter to the recommended default of `43200`, which is 12 hours. Setting this value to `0` prevents the cache from being purged, with the keys staying in the cache indefinitely.

   The following is an example of these parameters:

```
NAE_IP.1=10.8.176.38
NAE_Port=8943
Protocol=ssl
Key_Store_Location=C:\\TrustStore\\IngrianJCE\\myTrustStore.jks
Symmetric_Key_Cache_Enabled=yes
Symmetric_Key_Cache_Expiry=43200
```

   Note: You might need to specify additional parameters, depending on how CipherTrust/KeySecure is configured for your environment. For more information, see the CipherTrust/KeySecure documentation.

To connect an OpenText Information Archive component to CipherTrust / KeySecure appliance after you install the system component:

If you decide to encrypt your passwords and want to use CipherTrust, complete the following procedure.

1. Complete the procedure in Using the password encryption utility to manually encrypt passwords and tokens.

2. Copy the IngrianNAE JAR file into the following directories, depending on which OpenText Information Archive component or components you are installing:

3. If you want to use CipherTrust/KeySecure for data cryptography, then on each instance of IA Server, do one of the following:

   - If you saved the setup configuration as a template, in a command prompt, run the `iasetup` command with your template and `--config.review=true` as flags. For example:

     – Windows: `<IA_ROOT>\bin\iasetup apply --config.review=true`

     – Linux: `<IA_ROOT>/bin/iasetup apply --config.review=true`

     For more information about saving the setup configuration as a template, see section 10.3.1 “Generate the OpenText Information Archive configuration” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*.

   - If you did not save the setup configuration as a template, in a command prompt, run the following command:

     – Windows: `<IA_ROOT>\bin\iasetup iaserver`

     – Linux: `<IA_ROOT>/bin/iasetup iaserver`

     Prompts then appear for you to provide all IA Server settings.

     Caution

     Apart from the settings relevant to setting up CipherTrust/KeySecure for data cryptography, you should specify the same settings that you used previously.

     The following prompts are relevant for setting up CipherTrust/KeySecure for data cryptography:

     – `Enable use of Gemalto for data encryption: (y/N)`

       Type `y`.

     – `Gemalto Safenet Client properties file:`

       Specify the location of the properties file (for example, `<SHARED_RESOURCES_DIR>/gemalto/IngrianNAE.properties`).

4. Restart IA Server.

Chapter 4

Cryptography objects

Cryptography objects (also known as crypto objects or cryptography objects) contain parameters that are used for the encryption and decryption of data (for example, key size and algorithm). Instances of crypto objects are used as a reference by other configuration objects (for example, a database crypto or holding crypto) to provide encryption and decryption parameters. The database object type for a crypto object is `CryptoObject`.

4.1 Default crypto object

A default crypto object configures cryptography for the secret keys that OpenText Information Archive uses to encrypt and decrypt sensitive data in the system data database (such as passwords for databases and credentials for remote storage systems) and search results. It uses the crypto keystore.

OpenText Information Archive refers to the default crypto object through its application data cryptography object, which is a system-level object.

Caution

You must not modify or delete the default crypto object or the application data cryptography object once they are in use. If you do, then the passwords that were encrypted using the original parameters cannot be decrypted with the new parameters.

The default crypto object and application data cryptography object are created when OpenText Information Archive starts for the first time.

The default crypto object is visible in the **Administration > Encryption** tab in the IA Web App.

Use the IA Shell command `set-as-default` to set the context representing a crypto object to be the default.

There is a REST API for switching the default crypto object in the application data cryptography object (for example, if you decide that you require stronger encryption going forward). In that case, the original default crypto object continues to exist as a crypto object because it might still be needed (for example, to decrypt existing passwords). The new default crypto object is used to configure cryptography for the following:

- Sensitive data in new system data.
- New search results.

- Sensitive data in existing system data whenever that data is modified and saved. For example, if you change the name of an `RdbDataNode` object, then on save, OpenText Information Archive uses the new default crypto object to encrypt the superuser password.

4.2 Configuring crypto objects

Crypto objects have the following parameters. For more information about which combinations of settings are supported, see Cryptography providers, algorithms, modes, and padding for various ciphers.

Parameter	Type	Description	Mandatory	Label in UI
name	String	The name of the crypto object.	Yes	Name
securityProvider	String	The security provider used for encryption. Possible values are: • Bouncy Castle • Gemalto • SunJCE	Yes	Security provider
keySize	Integer	The size of the key used for encryption. Possible values are: • 128 • 192 • 256	Yes	Key size
encryption Algorithm	String	The algorithm used to encrypt data. Currently, the only possible value is AES.	Yes	Encryption algorithm
encryptionMode	Enumerated type	The algorithm used as part of encryption. Possible values are: • Bouncy Castle: CBC, CFB, CCM, GCM, GCF, GOFB, EAX, OCB, OFB, OpenPGPCG, and SIC. • Gemalto: CBC and ECB • SunJCE: CBC, CFB, CFB8, CFB16, CFB32, CFB64, CFB128, CTR, CTS, ECB, OFB, OFB8, OFB16, OFB32, OFB64, OFB128, and PCBC.	Yes	Encryption mode

Parameter	Type	Description	Mandatory	Label in UI
paddingScheme	String	The schema used to pad the encryption. Available padding schemes depend on the security provider. Possible values are: • Bouncy Castle: – ISO10126d2Padding – ISO7816d4Padding – PKCS7Padding – X932Padding – ZeroBytePadding • Gemalto: – NOPADDING – PKCS5PADDING • SunJCE: – ISO10126PADDING – NOPADDING – PKCS5PADDING	Yes	Padding
InUse	Boolean	Whether the crypto object is currently being used. You can only edit or delete a crypto object if it is not in use.	Transient (calculated by OpenText Information Archive)	In Use
naeCrypto-Configuration	NAECrypto-Configuration	The configuration required for crypto objects that use Gemalto as a security provider. It is not applicable to any other security provider.	Yes, if you are using Gemalto as a security provider	Network-Attached Encryption (NAE) Configuration

The NAE crypto configuration for Gemalto consists of the following additional parameters:

Parameter	Type	Description	Mandatory	Label in UI
name	String	The name of the NAE crypto configuration. This parameter is optional.	No	Configuration Name

Parameter	Type	Description	Mandatory	Label in UI
userName	String	The name of the user account that is configured on Gemalto appliance. When OpenText Information Archive connects to the Gemalto appliance to create a key, this user is the registered owner of the key.	Yes	User Name
password	String	The password for the user account that is configured on the Gemalto appliance.	Yes	Password
group	String	The name of the group on the Gemalto appliance. The Gemalto appliance allows the creation of both user accounts and groups, and the name of the group is required to set appropriate permissions on the secret keys that are generated.	Yes	Group Name
clientCertificateAlias	String	The name of the certificate alias that OpenText Information Archive presents to the Gemalto appliance to establish an SSL connection. The certificate must be in the truststore that is specified in the `Key_Store_Location` parameter in the `IngrianNAE.properties` file. For more information about the `IngrianNAE.properties` file, see Connecting system components to CipherTrust/KeySecure appliance.	Yes	Client Certificate Alias

Parameter	Type	Description	Mandatory	Label in UI
keyStorePassword	String	The password to the truststore that is specified in the `Key_Store_Location` parameter in the `IngrianNAE.properties` file.	Yes	Keystore Password

4.3 Managing crypto objects

You can create, read, update, or delete a crypto object (CRUD operations) using the following tools:

1. *IA Web App*: you can create, read, update, or delete a crypto object.
2. *Declarative configuration*: you can define a crypto object, which is then created when you import the application into OpenText Information Archive.
3. *IA Shell*: you can create, read, update, or delete a crypto object.
4. *REST*: you can create, read, update, or delete a crypto object with any REST client.

4.3.1 Managing crypto objects with IA Web App

A user with either the Administrator or Developer role can create, read, update, or delete a crypto object with IA Web App. You cannot edit or delete a crypto object if it is in use or referenced by any other object. In such a case, the crypto object is marked as **In Use** in the IA Web App.

The list of Cryptography Objects also indicates whether a specific object is the default object, An information icon appears after the name of the Cryptography Object if it is the default object. Furthermore, the information in the side panel also contains the **Default** field, which indicates whether the selected object is the default object.

To create a crypto object:

1. In IA Web App, go to the **Administration > Encryption** page.
2. Click the **+** button.
3. Enter the following information:
   - Name
   - Security provider
   - Key size
   - Encryption algorithm
   - Encryption mode

- Padding

For more information about possible values, see Cryptography objects.

4. If the security provider is Gemalto, specify the complete NAE crypto configuration.
5. Click the **Create** button.

   The crypto object is now listed in the Encryption table on the Encryption tab.

To find a crypto object:

1. In IA Web App, go to the **Administration > Encryption** page.
2. Click **Turn filter On** to activate the column filtering.
3. Use one of the following options:

   - **Match All Filters**: Enter criteria in each of the filters available on the **Encryption** tab: **Name**, **Security Provider**, **Key Size**, **Encryption Algorithm**, **Encryption Mode**, and **Padding**.
   - **Match Any Filter**: Enter criteria in any of the filters available on the **Encryption** tab.

To edit a crypto object:

1. In IA Web App, go to the **Administration > Encryption** page.
2. In the Encryption table, select the row for the crypto object that you want to edit.
3. In the right panel, click the **Edit item** icon 

   Note: The **Edit item** icon only appears if the crypto object is not in use.

To delete a crypto object:

1. In IA Web App, go to the **Administration > Encryption** page.
2. In the Encryption table, select the row for the crypto object that you want to delete.
3. Click the **Delete item** icon  at the end of the row.

   Note: The **Delete item** icon only appears if the crypto object is not in use.

4.3.2 Managing crypto objects with declarative configuration

You can define a new crypto object in a declarative configuration (DC). The crypto object is then created when you import the application into OpenText Information Archive. You must set the `configure` parameter to `create`, so the definition begins as follows:

```
cryptoObject:
  name: <NAME>
  configure: create
```

The rest of the `cryptoObject` definition consists of the usual parameters for a crypto object. For more information, see Cryptography objects.

You can view an example in the `<IA_ROOT>/examples/applications/PhoneCalls/config/configuration.yml` file. The relevant section is the `cryptoObject` section.

The following example illustrates some Gemalto SafeNet KeySecure specific parameters:

```
cryptoObject:
  name: Tresor-crypto
  configure: create
  paddingScheme: PKCS5Padding
  encryptionMode: CBC
  encryptionAlgorithm: AES
  keySize: 256
  securityProvider: Gemalto
  naeCryptoConfiguration:
    name: someName
    userName: connie
    password: s3cr3t123
    group: developer
    clientCertificateAlias: localclient
    keyStorePassword: abcdefg
```

You can also specify an existing crypto object in a declarative configuration. You must set the `configure` parameter to `use existing`, so the definition begins as follows:

```
cryptoObject:
  name: <NAME>
  configure: use existing
```

If you are using an existing crypto object, you do not need to specify the additional parameters in the rest of the `cryptoObject` definition. Only the name and `use existing` are required because the crypto object already exists and OpenText Information Archive locates it by name. To use an existing crypto object, the crypto object must exist during ingestion time.

4.3.2.1 Encrypting event context

Event or mixed retention policies can only be applied using a job, and can either be rule-based or not. When applying these types of retention policies, the job is typically configured to use a specific column as the context for the event. If the information is for a specific user, it may be considered sensitive information and should be encrypted, specifically for the event object.

The option to encrypt an event context can be done using declarative configuration (DC). The PhoneCallsGranular application has an example on how to enable the option (see `PhoneCallsGranular/config/application-config/configuration.yml` directory):

```
eventCrypto:
  name: PhoneCallsGranular-crypto
  cryptoObject: Demo-crypto
  application: PhoneCallsGranular
# END OF CRYPTO
```

By default, an event context is not encrypted and only necessary if the application is using either mixed or event retention policies.

An encryption object (`eventCrypto`) allows you to encrypt the sensitive information.

Caution

This is an optional configuration and meant only to encrypt sensitive information. Once you encrypt an event context for an application, it cannot be undone.

Do not reimport the application configuration (using DC) and change the name of the eventCrypto, as the system only permits one instance per application.

By default, the Event Crypto audit is enabled to track when an event context is created. Whether or not this event is enabled can be changed from the Administration > Audit tab. The audit's event category is Provisioning events and can be set for all applications or a single application.

4.3.3 Managing crypto objects with IA Shell

You can manage crypto objects using the following IA Shell commands:

Command	Description
`configure`	Create a crypto object
`cd /crypto-objects`	Retrieve the list of crypto objects
`update`	Update a crypto object
`delete`	Delete a crypto object

Command	Description
`set-as-default`	Sets the default crypto object. Refer to Using the `set-as-default` command for more information.

For more information about these commands and their parameters, see section 2 "IA Shell commands" in *OpenText Information Archive - IA Shell Guide (EARCORE-ARE)*.

4.3.3.1 Using the set-as-default command

You cannot use the `set-as-default` command if the crypto object is already the default. Prior to running the command, you can verify if the crypto object is already the default:

```
iashell> cat Trades-crypto
 Object:
 version: 1
 name: Trades-crypto
 applications: Trades
 createdBy: sue@iacustomer.com
 createdDate: 2022-06-17T20:00:59.131Z
 default: true
 encryptionAlgorithm: AES
 encryptionMode: CBC
 id: 178725f3-0944-4db1-a10e-e18e570d8e71
 inUse: true
 keySize: 256
 lastModifiedBy: sue@iacustomer.com
 lastModifiedDate: 2022-06-17T20:00:59.131Z
 naeCryptoConfiguration: null
 paddingScheme: PKCS5PADDING
 securityProvider: SunJCE
```

The following are the options for the `set-as-default` command:

- `[--d]`: Command context path
- `[--id --last]`: An ID value (self link) of a crypto object or its index in the result of the last `select` command.

The following is an example of the `set-as-default` command:

```
iashell>cd /crypto-objects/PhoneCalls-crypto
iashell>set-as-default
Set as default PhoneCalls-crypto
OK
```

4.3.4 Managing crypto objects with REST

You can perform a REST call through a link in the crypto object, which allows you to configure the crypto object to be used for application data encryption. Application data that was previously encrypted will not be re-encrypted because this data continues to use the crypto object that was used for its original encryption.

For more information about call methods to create, read, update, or delete a crypto object, see *OpenText Information Archive REST API Developers Guide* on support.opentext.com (https://support.opentext.com/).

Furthermore, the `set-as-default` command can be used to sets the context representing a crypto object to be the default.

Chapter 5

Configuring encryption for table-based application

Depending on whether an application is table-based or SIP-based, you should apply different configurations to encrypt structured and unstructured data during ingestion, and to decrypt data during the search.

This chapter covers how to configure encryption for table-based applications.

5.1 Encryption process

The encryption and decryption process is used throughout the life of a table archive, to do the following:

- Encrypt individual table column data during ingestion
- Encrypt the binary content during ingestion
- Query the table and decrypt result columns that are encrypted in the persistence layer

5.1.1 Ingestion

Based on the metadata file, business data is ingested with or without encryption. For more information about how to configure the metadata file for encryption, see Metadata file.

5.1.2 Search and data decryption

When individual columns have been encrypted, it is not possible to directly query over them because only the encrypted value and a hash of the original plain-text value is persisted in the database. It is, however, possible to prepare the SQL query to allow exact matches on the hash in combination with a post-process comparison against the decrypted value.

To do this, it is required for the SQL query parameter variable to be prefixed with `crypto:`, indicating that the provided value needs to be hashed before substitution.

By default, when "selecting" encrypted columns, the query returns the encrypted value. For the search result to decrypt such values, it is necessary to indicate that the to-be-returned column is encrypted by also prefixing it with `crypto:`. This results in the returned value being decrypted in a post-processing step before being persisted in the temporary search result.

It is also possible to re-encrypt any individual column being persisted in the search result, regardless of whether the original source column was encrypted, by marking the returned search result columns as such in the ResultMaster.

For an example, refer to the Baseball example application where the MASTER table has a column called NAMEFIRST that is marked to be encrypted.The following sample SQL query demonstrates the before-mentioned mechanisms to deal with this encrypted column:

```
SELECT IA_ROWID, NAMELAST as "lastName", crypto:NAMEFIRST as "firstName"
FROM BASEBALL.MASTER WHERE 1=1
% AND NAMEFIRST = $crypto:firstName%
% AND NAMELAST = $lastName%
<encrypt>true</encrypt>
```

The breakdown for this SQL query is as follows:

```
SELECT IA_ROWID, NAMELAST as “lastName”, crypto:NAMEFIRST as “firstName”
```

Identify all of the columns to return in the search result:

```
crypto:NAMEFIRST
```

The column `NAMEFIRST` is encrypted and needs to be decrypted before being returned as part of the search result as `firstName`. Rename the column NAMEFIRST as `firstName`in the search result

```
WHERE 1=1
```

Required because both `where` clauses are optional. This results in there always being at least one `where` clause to be evaluated, but never filters out any rows unintentionally.

```
% AND NAMEFIRST = $crypto:firstName%
```

The %...% indicates this is an optional clause only to be included when the variable within (identified by the `$` prefix) is provided a value. The `$` prefix indicates that what follows is the name of a search query parameter.

```
crypto:firstName
```

The `crypto:` prefix indicates the value for this variable needs to be hashed because the column to which it applies is encrypted. Attempting to compare the plain-text value with an encrypted column, or the hash with a non-encrypted column will never match.

`firstName` is the name of the variable provided to the query for which the value will be substituted when running the resulting SQL query

5.2 Table configuration resources

There are two types of resources that you use to configure encryption for table-based applications:

- Metadata file
- Database crypto

5.2.1 Metadata file

The metadata file is an XML file that describes the data to be ingested. To define which table columns are to be encrypted, you add an `encrypt` element to the column element:

```
<encrypt>true</encrypt>
```

You can view an example in the `<IA_ROOT>/examples/applications/Baseball/config/data-model-config/database-BaseballSqlDb-00001.xml` file.

5.2.2 Database crypto

The database crypto is a system-level object that contains settings to be used when encrypting data for a table application. The object type is `DatabaseCrypto`.

You can find an example object definition for a database crypto in the `<IA_ROOT>/examples/applications/Baseball/config/application-config/data-model-config/configuration.yml` file.

With the attributes of the database crypto object definition, you can specify if structured and/or unstructured content encryption is enabled or disabled. If enabled, you must specify the name of the crypto object used for the encryption. It is possible to have different crypto objects for structured and unstructured content.

Chapter 6

Configuring encryption for SIP-based applications

Depending on whether an application is table-based or SIP-based, you should apply different configurations to encrypt structured and unstructured data during ingestion, and to decrypt data during the search.

This chapter covers how to configure encryption for SIP-based applications.

6.1 Encryption process

The encryption and decryption process is used throughout the life of a SIP archive, to do the following:

- Encrypt the SIP package during reception
- Encrypt the binary content during ingestion
- Encrypt a subset of metadata during ingestion
- Decrypt and query the metadata during the search
- Decrypt the binary content during content retrieval

6.1.1 Reception

At the reception level, the SIP package is encrypted if:

- The holding crypto is set to encrypt the SIP package
- The holding crypto specifies the crypto object to use to encrypt the SIP package

For more information about how to configure the holding crypto for encryption, see Holding crypto.

Once encrypted, the format of the corresponding rendition is `sip_zip_crypto`. The crypto object, `keyId`, and initialization vector (IV) used to encrypt the file are stored in the AIP. The `keyId` is the identifier that OpenText Information Archive uses to request the key from the security provider that is used for symmetric encryption.

6.1.2 Ingestion

The following processors are used to manage encryption as follows:

- `SipDecryptProcessor`: Decrypt the SIP, if it was encrypted at the reception level.
- `LucenePdiParserProcessor`:
  - Encrypt and hash the metadata if:
    - The path of the element is set in the PDI crypto configuration.
    - The holding crypto enables the PDI encryption.
    - The crypto object is defined in the holding crypto for PDI.
  - Encrypted metadata and the hash value are encoded in base64.
  - The result of the encryption should appear like the following:

Type of Value	Before	After
Text Value	`<CustomerID>value</ CustomerID>`	`<CustomerID ns1:hash="base64 encoded hash">base64 encoded encrypted value </ CustomerID>`
Attribute	`<Customer ID="value" />`	`<Customer ns1:hash="base64 encoded hash" ID="base 64 encoded encrypted value" />`

  - The `keyId`, IV, hash salt, crypto encoding and crypto object used are stored in the AIP.

    The hash salt is a variable that strengthens cryptography. When OpenText Information Archive encrypts a value from the PDI file, it generates a hash to be able to search for the encrypted value without decrypting the whole base. To avoid having the same hash values for different AIPs, OpenText Information Archive uses a different hash salt for each AIP.
- `PdiFileEncryptProcessor`:
  - Encrypt the PDI file, given the following:
    - The holding is set to keep this file after ingestion.
    - The holding crypto enables the PDI encryption.
    - The crypto object is defined in the holding crypto for the PDI.
  - Store the keyId, IV and crypto object used in the AIP.
  - The format for the rendition of the encrypted file is pdi_xml_gzip_crypto.
- LuceneCiContainerProcessor:
  - Encrypt the contents of the AIP, given the following:

    - The encryption is enabled for the content in the PDI configuration.
    - The holding crypto enables the content information (CI) encryption.
    - The crypto object is defined in the holding crypto for CI.
  - Store the keyId, IV and crypto object used in the AIP.
  - Update the table of the RI for the content that is to be encrypted.

6.2 SIP configuration resources

There are five types of resources that are used for configuring encryption for SIP-based applications:

- Holding crypto
- PDI
- PDI crypto
- AIC
- Query

6.2.1 Holding crypto

A holding crypto is a system-level object that contains encryption settings to be used during data ingestion. The object type is `HoldingCrypto`. This object allows an Administrator to enable or disable encryption on a SIP, including its PDI file and content files. Search on encrypted values and decryption of encrypted values in search results are handled by the AIC and Query object.

A holding crypto contains the following configurations:

- The PDI crypto configuration to be applied during ingestion
- The encryption configuration for the SIP, PDI file, and content files

A holding crypto has the following parameters. You can use this table as a reference if you want to modify your configuration outside of the holding wizard.

Parameter	Type	Description	Mandatory
id	UUID	A unique identifier.	Yes
name	String	The name of the holding crypto.	Yes
application	Application	The name of the application.	Yes
holding	Holding	The name of the holding.	Yes

Parameter		Type	Description	Mandatory
pdis	schema	String	The PDI schema that the PDI crypto configuration is applied to.	Yes
	pdiCrypto	PdiCrypto	The PDI crypto configuration to apply.	Yes
sip	cryptoEnabled	Boolean	Whether SIP rendition encryption is enabled.	No
	cryptoObject	CryptoObject	The name of the crypto object to use for SIP rendition encryption.	No
pdi	cryptoEnabled	Boolean	Whether PDI metadata and rendition encryption is enabled.	Yes
	cryptoObject	CryptoObject	Name of the crypto object to use for PDI metadata and rendition encryption.	Yes
ci	cryptoEnabled	Boolean	Whether encryption for unstructured content is enabled.	Yes
	cryptoObject	CryptoObject	The name of the crypto object to use for the encryption of unstructured content.	Yes

6.2.2 PDI

The PDI is the configuration for the ingestion. It especially describes the way unstructured contents of the AIP are managed. An Administrator defines here if unstructured content should be encrypted.

The PDI parameters used for encryption are:

Parameter	Type	Description	Mandatory
cis encrypt	Boolean	Whether encryption for this CI unstructured content is enabled.	Yes

6.2.3 PDI crypto

A PDI crypto is an ingestion configuration that overrides the standard PDI ingestion configuration of a holding for all ingestion processors related to encryption.

The object type is `PdiCrypto`. This object allows an Administrator to define the metadata to be encrypted and redefine the indexes, if the encrypted metadata is indexed

Indexes were previously defined in the PDI configuration. The goal of index redefinition in the PDI crypto is to redefine indexes where encryption changes the type of the data, from an XML point of view. For example, if encryption is not enabled, a date is indexed as a date, such as 2018-05-20. If encryption is enabled, the date is encrypted as a string, such as `QOIJF0329Y390HOFZJDE=/`.

Note: If `PDICrypto.encryptedPaths` configuration is updated after the installation, and some AIPs have already been ingested, the libraries corresponding to those AIPs will be considered as out-of-sync by the system. Therefore, the user will not be able to launch synchronous searches targeting those AIPs. In this case, the system will propose to launch a background search instead. Doing so will re-index the library during the background search run. All out-of-sync libraries can also be re-synchronized using the Post Ingest Processing job. For more information, see section 3.7.18 “Post Ingest Processing job” in *OpenText Information Archive - Administration Guide (EARCORE-AGD)*.

A PDI crypto has the following parameters:

Parameter	Type	Description	Mandatory
id	UUID	A unique identifier.	Yes
name	String	The name of the PDI crypto.	Yes
application	Application	In the database, the UUID of the `Application` data object. In the installation files, the name of the `Application` object.	Yes

Parameter	Type	Description	Mandatory
encryptedSchema	String	Virtual schema given to the AIP after encryption (allows Administrator to make a difference between encrypted and unencrypted AIPs, or discriminate evolution in the encryption configuration).	No

Parameter		Type	Description	Mandatory
encryptedPaths	path	String	Path of the element to encrypt	No
	type	Type	Type of the element to encrypt	No

The PDI crypto has the following limitations:

- A text value and an attribute cannot be encrypted on the same element.
- Only one attribute can be encrypted by an element.
- The namespace must be set for every element.
- You should set the namespace for every attribute, whether or not it is set in the PDI file. If the attribute is not qualified in the PDI file, then OpenText Information Archive assumes that the attribute's name is qualified by the namespace of the parent element.

Note: Encryption of any partition key is not supported.

6.2.4 AIC configuration

If PDI encryption is enabled, optionally, you can update the AIC configuration to allow specified groups to search for and view encrypted data. A member of a specified group can input a clear value, and then OpenText Information Archive hashes the value and compare it to the hash of the encrypted value. If there is a match, the result is returned.

- `pdiCryptoQueryGroup`: Specifies the role or roles that can use encrypted search criteria in a query. If blank, any role can use encrypted search criteria.
- `pdiCryptoAccessGroup`: Specifies the role or roles that can view decrypted data for search results. If blank, any role can view the decrypted data.

Criteria that can be encrypted in the PDI file should be defined with the type "hashed" in the AIC.

If an unauthorized user attempts to search for and view encrypted data, OpenText Information Archive responds as follows:

- If the user is not authorized with `pdiCryptoQueryGroup`, the server responds with a: `401 Unauthorized` response code.
- If the user is not authorized with `pdiCryptoAccessGroup`, any encrypted data in the search result is not decrypted. The encrypted data is displayed as *****.

6.2.5 Query configuration

If PDI encryption is enabled, you must update the query configuration to specify how the encryption schema should be handled. Update the following attributes for the Query object in the declarative configuration:

- In the `libraryPdiConfigs` section, add the following parameters:

Parameter	Description	Example
schema	The PDI encrypted schema.	urn:eas-samples:en:xsd:phonecalls.1.0:crypto
entityPath	The root element of an entity in the PDI.	/n:Calls/n:Call

operands	The list of operands as in a standard `libraryPdiConfigs` parameter, except the encrypted elements should be defined as follows:
	`- name: <NAME_OF_OPERAND>`
	`  path: ${<PATH_OF_OPERAND>}/@ia-pdi:hash`
	`  type: HASHED`
		`- name: CustomerLastName`
		`  path: n:CustomerLastName/@ia-pdi:hash`
		`  type: HASHED`

The examples above are from the `<IA_ROOT>/examples/applications/PhoneCalls/config/searches/configuration.yml` file.

6.3 Allotting encryption keys

For SIP ingestion, you can configure the key allotment scheme for OpenText Information Archive so that it switches encryption keys after one of the following periods:

- A day
- A week
- A month
- A quarter (three months)
- Semi-annually (six months)
- Annually (twelve months)
- None (the default, which means that the key is not switched)

Key allotment does not enable a rotation of keys or the updating of an existing key. It uses a new key for subsequent data ingestion, without changing any of the keys for data that has already been ingested.

You can limit the scope of the key allotment to either an application (the default) or a holding. If there is only one holding in an application, then setting the scope to application or holding has the same effect.

There are two places where you can configure key allotment: in the holding wizard or in a declarative configuration.

Note: Do not use a key allotment scheme if you are storing your keystores on a filesystem. The key allotment scheme is allowed only if the keystores are stored in a database or in Gemalto SafeNet KeySecure.

To configure the key allotment scheme in the holding wizard:

1. In the **Ingestion > Encryption** step of the holding wizard, specify the following settings:
   - **Key Scope**
   - **Key Allotment Scheme**
2. Specify the other settings in the holding wizard that you want.

To configure the key allotment scheme in a declarative configuration:

1. In a text editor, open the application's `config/configuration.yml` file (for example, `<IA_ROOT>/examples/applications/PhoneCalls/config/application-config/configuration.yml`).
2. Change the `holdingCrypto.keyIdScope` parameter to one of the following values:
   - `APPLICATION`
   - `HOLDING`
3. Change the `holdingCrypto.keyIdAllotmentScheme` parameter to one of the following values:
   - `DAY`
   - `WEEK`
   - `MONTH`
   - `QUARTER`
   - `SEMIANNUAL`
   - `ANNUAL`
   - `NONE`

4. For the new settings to take effect, install the holding again to run an update.

Chapter 7

Encrypting component passwords and secret tokens in configuration files

For many organizations, storing component passwords and secret tokens in clear text (unencrypted) is a significant security risk. For example, many finance and health care organizations are governed by strict security regulations. You should encrypt all of the passwords and secrets that are stored in the configuration files of each of your OpenText Information Archive components. This is recommended for all production configurations.

OpenText Information Archive includes support to the install software to generate a password encryption configuration, but still relied on manually encrypting passwords. You can encrypt component passwords and secret tokens automatically using the install software.

When you use the install software to encrypt passwords, any passwords found in configuration files or provided for encryption are automatically encrypted using the appropriate settings. However, it might still be useful for you to be able to encrypt passwords manually and know how to verify that password encryption is configured properly.

Note: Encrypting passwords using the install software is strongly recommended over the manual method. You can only perform the manual method after you have already encrypted passwords using the install software.

Caution

Before you encrypt any passwords or secrets, you should make sure that you take note of all passwords and secrets, including the keystore password. If no one knows a password or secret and the system is subsequently encrypted, there is no easy or convenient way to get back the unencrypted password or secret.

7.1 Encrypting passwords using the install software

During your initial installation of OpenText Information Archive, you can choose to encrypt the component passwords and secret tokens in configuration files.

If you choose not to encrypt passwords, but later decide that you want to do so, you can run the install software again and encrypt the passwords by running the following command:

```
<IA_ROOT>/bin/iasetup encrypt-passwords
```

This is the recommended method for encrypting passwords. For more information, see section 8.4 “Encrypting passwords during installation” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*.

If you want to encrypt passwords using CipherTrust/KeySecure, you must first connect the OpenText Information Archive components to CipherTrust/KeySecure. For more information, see Connecting system components to CipherTrust/KeySecure appliance.

7.2 Encrypting passwords manually

After you install OpenText Information Archive, including choosing to encrypt passwords with the install software, you can encrypt the passwords again manually. The manual method is not recommended for encrypting passwords. If you want to re-encrypt the passwords or change your password encryption settings, you should use the install software to do so instead.

To encrypt passwords and secrets manually in configuration files, there are three tasks that you must perform:

1. Configure how encryption is performed and encrypt your passwords and secrets.
2. Configure settings and store the encrypted passwords and secrets in the configuration files.
3. Restart the components and enter the keystore password.

Notes

- You must do the steps above for each instance of an OpenText Information Archive component.
- You cannot mix encrypted and unencrypted passwords in a configuration file. If you encrypt one or more passwords or secrets in a configuration file, then you must encrypt all of them.
- By encrypting passwords using the `iasetup` script, most of this is automated and you just need to run it on each node and restart each component afterwards. If you do not use the `iasetup` script, you must do all of this manually.

- You can also follow the steps above if you want to change any encryption settings or re-encrypt any passwords or secrets.

7.2.1 Encrypting database passwords in configuration files for applications

By default, applications store database passwords in clear text (unencrypted) in configuration files. An application's configuration file (for example, `<IA_ROOT>/config/iashell/default.properties`) specifies several passwords for a database. Refer to the example below.

- `rdbDataNode.password`
- `rdbDatabase.password`

When application's configuration file, for example applications, specifies these passwords, they use references to properties. For example:

```
superUserPassword: ${rdbDataNode.password:}
```

You can specify properties or the actual password values.

When you install OpenText Information Archive, the `<IA_ROOT>/config/iashell/default.properties` file is automatically generated. This file includes definitions for the properties in the application's configuration files, for example applications. You can override any settings in the `default.properties` file by copying the parameter into the `config/configuration.properties` file for an application and changing the value there.

You can encrypt the passwords instead of specifying the actual password values in clear text.

When you rely on the install software to encrypt passwords, only the database related passwords in the `default.properties` file are automatically encrypted. Any other passwords in the `default.properties` file (for example, storage related passwords) as well as any passwords in the `configuration.yml` files for applications still need to be manually encrypted.

7.2.2 Password encryption utility

You can use OpenText Information Archive's password encryption utility to encrypt any passwords with the settings that you provide. You can find the utility in the `<IA_ROOT>/bin` directory:

- Windows: `password-encrypt.bat`
- Linux: `password-encrypt`

You configure the password encryption utility by specifying the following parameters in the `<IA_ROOT>/config/password-encrypt/application.yml` file.

Notes

- Possible values for the parameters are the same as for crypto objects. For more information about possible values, see Configuring crypto objects.
- For each component that you want to encrypt passwords for, you must use the same values for corresponding parameters. When you encrypt passwords using the install software, this is taken care of automatically. However, when you make manual changes to the password encryption configuration, you must manually propagate the same changes to all affected components.

  Failing to do so, and thus using different values across components, might result in both the install software as well as individual components failing to properly decrypt certain passwords.

`securityProvider`
The security provider used for encryption (`SunJCE`, `Bouncy Castle`, or `Gemalto`).

`encryptionAlgorithm`
The algorithm used for encryption (`AES`).

`encryptionMode`
The encryption mode used (for example, `CBC`).

`paddingScheme`
The padding scheme used during encryption (for example, `PKCS5PADDING`).

`keyStoreType`
The type of keystore to use (for example, `jceks`).

`keyStorePath`
The absolute path of the keystore where the encryption key is stored. You should store the keystore in a separate directory structure from `<IA_ROOT>`, for easier file preservation during upgrade. For example:

```
keyStorePath: C:/keystores/keystore.jceks
```

If you are running OpenText Information Archive components on separate computers, then the location of the keystore should be a network location that is accessible to all of the computers that host OpenText Information Archive components.

When you run the `password-encrypt` script for the first time, the script generates the keystore in the location specified by `keyStoreLocation`.

Should the keystore be overwritten or regenerated, or the settings are changed, any passwords encrypted using a different or older keystore or settings can no longer be decrypted. In this case, the plain text equivalents of the passwords need to be re-encrypted using the new keystore and settings

`keySize`
The size of the key being used for encryption (for example, `128`, `192`, or `256`).

`keyID`
The ID of the key being used for encryption. When it does not already exist in the keystore, it is created.

`gemaltoPropertiesFile`
The absolute path of the properties file for the CipherTrust/KeySecure appliance. For example:

```
gemaltoPropertiesFile: C:\infoarchive\config\iaserver\IngrianNAE.properties
```

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`gemaltoClientCertificate`
The name of the certificate alias that OpenText Information Archive presents to CipherTrust/KeySecure appliance to establish an SSL connection.

The properties file specifies a keystore location. The keystore might have multiple certificates, so this setting specifies the certificate.

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`gemaltoUsername`
The name of the user account that is configured on CipherTrust/KeySecure appliance.

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`gemaltoGroup`
The name of the group that the gemaltoUsername account belongs to on CipherTrust/KeySecure appliance. This group is used to set appropriate permissions on the keys created in CipherTrust/KeySecure appliance, under the user account's name.

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`noPromptStartup`
If you do not want to be prompted to enter the keystore password when you start up the password encryption utility or any of the OpenText Information Archive components, set this parameter to `true`.

The password encryption utility will then generate two additional files in the same directory where the keystore is created:

- `secretStore.uber`: This is a proprietary Bouncy Castle store, and it stores the keystore password in an encrypted manner.
- `creds`: This file contains the password for secretStore.uber in a masked form.

As long as these two files exist, the password encryption utility uses the passwords in the keystore when starting, even if you subsequently set `noPromptStartup` to `false`. You must also make sure that the password encryption utility can read the two files, and that they are stored in the same directory as the keystore.

Caution
You must protect access to the `secretStore.uber` and `creds` files.

7.2.3 Using the password encryption utility to manually encrypt passwords and tokens

If SunJCE or Bouncy Castle is your security provider, this functionality stores the key used for password encryption in a local keystore or a keystore on a network location. The keystore is file-based and it is up to you and your organization to protect it.

If Gemalto is your security provider, this functionality stores the key used for password encryption in CipherTrust/KeySecure appliance.

To encrypt your passwords and secrets:

This procedure encrypts all passwords and secrets for any OpenText Information Archive components installed on the current machine. This procedure needs to be repeated for each machine hosting components.

1. If you want to encrypt passwords, run one of the following commands:
   - Windows: `<IA_ROOT>\bin\iasetup password-encrypt`
   - Linux: `<IA_ROOT>/bin/iasetup password-encrypt`
2. The following prompts may appear. Any passwords requested will have to be entered twice to confirm.
   - You will be asked about the `Shared resources location`, which is important if you are configuring Gemalto.
   - For more information about the other questions, refer to Interview questions asked when encrypting passwords and secrets.

To decrypt your passwords and secrets:

This procedure decrypts all passwords and secrets for any OpenText Information Archive components installed on the current machine. The main reasons for decrypting passwords are because you require the plain text or need to change the encryption settings and re-encrypt the information.

1. Open a command prompt, go to the `<IA_ROOT>/bin` directory, and then run the `iasetup decrypt-passwords` script.
2. For more information about the other questions, refer to Interview questions asked when encrypting passwords and secrets.

To encrypt a single field representing sensitive information:

1. Open a command prompt, go to the `<IA_ROOT>/bin` directory, and then run the `password-encrypt` script.
2. The following prompts appear:
   a. `Enter password for encryption:`

Specify the password or secret that you want to encrypt. You will not see any letters or symbols appear on the screen. This lack of on-screen feedback for typed characters is done for security reasons.

b. `Verify password for encryption:`

Specify the same password or secret again.

c. `Enter KeyStore password:`

If you are running the `password-encrypt` script for the first time, then the password that you specify becomes the password for the keystore.

If you are not running the `password-encrypt` script for the first time, then you must specify the password that you set for the keystore when you ran the `password-encrypt` script for the first time.

d. If your security provider is Gemalto SafeNet KeySecure, you will also be prompted for the KeySecure password.

To configure settings and store encrypted passwords and secrets for IA Server:

1. In a text editor, open the `<IA_ROOT>/config/iaserver/application.yml` file.
2. In the `passwordEncryption` section, make sure that the parameters match the equivalent parameters in the `<IA_ROOT>/config/password-encrypt/application.yml` file. Note that some parameters might be specified differently (for example, with or without quotation marks). If these parameters do not already exist in the file, you must add them.

   The parameters are as follows:

   `keyStorePath`
   The absolute path of the keystore where the encryption key is stored. For example:

   ```
   keyStorePath: "C:/keystores/iaserver/keystore.jceks"
   ```

   If you are running OpenText Information Archive components on separate computers, then the location of the keystore should be a network location that is accessible to all of the computers that host OpenText Information Archive components.

   `keyStoreType`
   The type of keystore to use (for example `jceks`).

   `keyID`
   The ID of the key being used for encryption.

   `enabled`
   Set this value to `true`.

   `encryptionAlgorithm`
   The algorithm used for encryption (`AES`).

   `encryptionMode`
   The encryption mode used (for example, `CBC`).

`paddingScheme`
The padding scheme used during encryption (for example, `PKCS5PADDING`).

`securityProvider`
The security provider used for encryption (for example, `SunJCE`, `Bouncy Castle`, or `Gemalto`).

`keySize`
The size of the key being used for encryption (for example, `128`, `192`, or `256`).

`gemaltoPropertiesFile`
The absolute path of the properties file for Gemalto SafeNet KeySecure. For example:

```
gemaltoPropertiesFile: C:\infoarchive\config\iaserver\IngrianNAE.properties
```

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`gemaltoClientCertificate`
The name of the certificate alias that OpenText Information Archive presents to CipherTrust/KeySecure appliance to establish an SSL connection.

The properties file specifies a keystore location. The keystore might have multiple certificates, so this setting specifies the certificate.

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`gemaltoUsername`
The name of the user account that is configured on CipherTrust/KeySecure appliance.

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`gemaltoGroup`
The name of the group that the gemaltoUsername account belongs to on CipherTrust/KeySecure appliance. This group is used to set appropriate permissions on the keys created in CipherTrust/KeySecure appliance, under the user account's name.

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

3. For each password or secret, replace the unencrypted clear text of the password or secret with the encrypted password or secret.
4. Using a text editor, in each of the following files, replace the unencrypted clear text of each password or secret with the encrypted password or secret:
   - `<IA_ROOT>/config/iaserver/application-https.yml`
   - `<IA_ROOT>/config/iaserver/application-infoarchive.ias.profile.OTDS.yml`
5. Restart IA Server.
6. If prompted, enter the keystore password.

To configure settings and store encrypted passwords for IA Web App:

1. In a text editor, open the `<IA_ROOT>/config/iawebapp/application.yml` file.
2. In the `passwordEncryption` section, make sure that the settings match the equivalent settings in the `<IA_ROOT>/config/password-encrypt/application.yml` file. Note that some settings might be specified differently (for example, with or without quotation marks).

   The parameters are as follows:

   `keyStorePath`
   The absolute path of the keystore where the encryption key is stored. For example:

   ```
   keyStorePath: file:C:/keystores/iawebapp/keystore.jceks
   ```

   If you are running OpenText Information Archive components on separate computers, then the location of the keystore should be a network location that is accessible to all of the computers that host OpenText Information Archive components.

   `keyStoreType`
   The type of keystore to use (for example `jceks`).

   `keyID`
   The ID of the key being used for encryption.

   `enabled`
   Set this value to `true`.

   `encryptionAlgorithm`
   The algorithm used for encryption (for example, `AES`).

   `encryptionMode`
   The encryption mode used (for example, `CBC`).

   `paddingScheme`
   The padding scheme used during encryption (for example, `PKCS5PADDING`).

   `securityProvider`
   The security provider used for encryption (for example, `SunJCE`, `Bouncy Castle`, or `Gemalto`).

   `keySize`
   The size of the key being used for encryption (for example, `128`, `192`, or `256`).

   `gemaltoPropertiesFile`
   The absolute path of the properties file for CipherTrust/KeySecure appliance. For example:

   ```
   gemaltoPropertiesFileLocation: file:C:\infoarchive\config\iaserver
   \IngrianNAE.properties
   ```

   This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`gemaltoClientCertificate`
The name of the certificate alias that OpenText Information Archive presents to CipherTrust/KeySecure appliance to establish an SSL connection.

The properties file specifies a keystore location. The keystore might have multiple certificates, so this setting specifies the certificate.

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`gemaltoUsername`
The name of the user account that is configured on CipherTrust/KeySecure appliance.

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`gemaltoGroup`
The name of the group that the `gemaltoUsername` account belongs to on CipherTrust/KeySecure appliance. This group is used to set appropriate permissions on the keys created in CipherTrust/KeySecure appliance, under the user account's name.

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

3. For each password or secret, replace the unencrypted clear text of the password or secret with the encrypted password or secret.
4. Using a text editor, in each of the following files, replace the unencrypted clear text of each password or secret with the encrypted password or secret:
   - `<IA_ROOT>/config/iawebapp/application-CLIENTS.yml`
   - If you are using Microsoft Active Directory: `<IA_ROOT>/config/iawebapp/application-infoarchive.gateway.profile.AUTHENTICATION_ACTIVE_DIRECTORY.properties`
   - If you are using LDAP: `<IA_ROOT>/config/iawebapp/application-infoarchive.gateway.profile.AUTHENTICATION_EXTERNAL_LDAP.properties`
   - If you are using OTDS:
     - `<IA_ROOT>/config/iawebapp/application-infoarchive.gateway.profile.AUTHENTICATION_OTDS.yml`
     - `<IA_ROOT>/config/iawebapp/application-infoarchive.gateway.profile.OTDS.yml`
   - If you are using example user accounts: `<IA_ROOT>/config/iawebapp/application-infoarchive.gateway.profile.AUTHENTICATION_IN_MEMORY.properties`
5. Restart IA Web App.
6. If prompted, enter the keystore password.

To configure settings and store encrypted passwords for IA Shell:

1. In a text editor, open the `<IA_ROOT>/config/iashell/application.yml` file.
2. Locate the `passwordEncryption` section.
3. Make sure that the following settings match the equivalent settings in the `<IA_ROOT>/config/password-encrypt/application.yml` file. Note that some settings might be specified differently (for example, with or without quotation marks).

   The parameters are as follows:

   `enabled`
   Set this value to `true`.

   `keyStorePath`
   The absolute path of the keystore where the encryption key is stored. For example:

   ```
   keyStorePath: file:C:/keystores/iashell/keystore.jceks
   ```

   If you are running OpenText Information Archive components on separate computers, then the location of the keystore should be a network location that is accessible to all of the computers that host OpenText Information Archive components.

   `keyStoreType`
   The type of keystore to use (for example `jceks`).

   `keyID`
   The ID of the key being used for encryption.

   `encryptionAlgorithm`
   The algorithm used for encryption (for example, `AES`).

   `encryptionMode`
   The encryption mode used (for example, `CBC`).

   `paddingScheme`
   The padding scheme used during encryption (for example, `PKCS5PADDING`).

   `securityProvider`
   The security provider used for encryption (for example, `SunJCE`, `Bouncy Castle`, or `Gemalto`).

   `keySize`
   The size of the key being used for encryption (for example, `128`, `192`, or `256`).

   `gemaltoPropertiesFile`
   The absolute path of the properties file for CipherTrust/KeySecure appliance. For example:

   ```
   gemaltoPropertiesFile: file:C:/infoarchive/config/iaserver/
   IngrianNAE.properties
   ```

   This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`gemaltoClientCertificate`
The name of the certificate alias that OpenText Information Archive presents to CipherTrust/KeySecure appliance to establish an SSL connection.

The properties file specifies a keystore location. The keystore might have multiple certificates, so this setting specifies the certificate.

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`gemaltoUsername`
The name of the user account that is configured on CipherTrust/KeySecure appliance.

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

`gemaltoGroup`
The name of the group that the `gemaltoUsername` account belongs to on CipherTrust/KeySecure appliance. This group is used to set appropriate permissions on the keys created in CipherTrust/KeySecure appliance, under the user account's name.

This parameter is applicable only if `securityProvider` is set to `Gemalto`.

4. For each password or secret, replace the unencrypted clear text of the password or secret with the encrypted password or secret.

To use encrypted passwords for core IA Apps or customer IA Apps:

1. In a text editor, open the `<IA_ROOT>/config/iashell/default.properties` file.
2. Set the `credentialsAreEnctrypted` parameter to true.
3. Replace the unencrypted clear text of the following passwords with the encrypted passwords. Note that this is only required for table applications since SIP applications do not use PostgreSQL, so these four values are not specified.

   These are the values that you need to update:

   - `rdbDataNode.password`
   - `rdbDatabase.password`
   - `rdbDataNodes.connectionProperties.sslpassword`
   - `rdbDatabases.connectionProperties.sslpassword`

   Note: Note that the last two entries only need to be modified if you are using SSL and have decided to encrypt the `sslcert`.

7.2.4 Starting IA Server, IA Web App, and IA Shell without password input

You can configure OpenText Information Archive so that you are not prompted to enter the keystore password when you start IA Server, IA Web App, and IA Shell.

To start system components without password input:

1. Do one of the following:
   - When you are installing OpenText Information Archive, and you are prompted whether to `Enable no prompt startup`, type `y`.
   - For an OpenText Information Archive installation that does not have encrypted passwords, run the `iasetup` script with the `encrypt-passwords` option to encrypt the passwords, and when prompted, choose to enable no prompt startup.
   - For an OpenText Information Archive installation that already has encrypted passwords, run the `iasetup` script with the `--passwordEncryption.noPromptStartup=true` option.
   - If you want to encrypt passwords manually, when you are configuring the password encryption utility, set `noPromptStartup` to `true`, and then encrypt the passwords.

   Note: For more information about the OpenText Information Archive install software and its scripts, see section 13.7.1 “Scripts” in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*.

2. Make sure that the keystore, the `secretStore.uber` file, and the `creds` file are accessible to all of the OpenText Information Archive components (for example, on a network filesystem or copied to each of the computers that host the OpenText Information Archive components).

   Notes
   - For each OpenText Information Archive component, the `keyStorePath` parameters must point to the correct location for these files. Update the parameters if you have chosen to move the files.
   - If you want to be prompted for the password again, delete the `secretStore.uber` and `creds` files.

Chapter 8

Azure storage encryption using an HSM vault

Before deploying OpenText Information Archive (or any other application) on a cluster, you need to make decisions on the underlying storage configuration. In the Azure cloud, that includes the configuration of storage encryption. This chapter discusses one aspect of storage encryption, which is the management of its key(s). Azure lets you choose between having them managed for you entirely or keeping them under user/customer control. With user-managed keys, you can choose between software and hardware-backed vaults. The latter are called "HSM" (Hardware Security Module) vaults, providing enhanced security for cryptographic keys through physical hardware protection. Because of the hardware aspect, HSM vaults are considerably more expensive than software vaults.

This following sections of this chapter discuss the setup of an HSM vault itself, including the creation of a key. This is followed by a short section on setting up OpenText Information Archive for "using" the HSM vault. Finally, some migration aspects are covered if you are coming from a deployment without an HSM vault but want to migrate to HSM vault-backed storage.

8.1 Creating an HSM vault

To learn more about vault creation, see the Azure document titled *Quickstart: Provision and activate a Managed HSM using Azure CLI*.

To create an HSM vault:

1. Determine the ID of the initial admin of the vault (this can also be a managed identity):

```
$ oid=$(az ad signed-in-user show --query id -o tsv)

$ echo $oid
4606d28a-71c7-4c93-8fe4-167a3fd93422
```

2. Using this ID, create the vault (in our example, it is called `testhsm`).

   The following information is required:

   a. Provide your resource group and location.

   b. Provide the number of days the vault will be retained after you delete it.

      Tip: As HSM vaults are expensive, when testing, set it to the minimum value of 7 days.

3. Enable purge protection. Azure will not let you use the vault for a storage account (see below) if you do not enable this, as it protects you from throwing keys away that are in use.

Example 8-1: Command to enable pure protection

```
$ az keyvault create --hsm-name "testhsm" --resource-group "rg-infoarchive-
cluster_westus" --location "westus" --administrators $oid --retention-days 7 --
enable-purge-protection true
{
  "id": "/subscriptions/59dfc29c-c8dd-4293-823d-e82a93cce062/resourceGroups/rg-
infoarchive-cluster_westus/providers/Microsoft.KeyVault/managedHSMs/
testhsm",
  "identity": null,
  "location": "westus",
  "name": "testhsm",
  "properties": {
    "createMode": null,
    "enablePurgeProtection": false,
    "enableSoftDelete": true,
    "hsmUri": "https://testhsm.managedhsm.azure.net/",
    "initialAdminObjectIds": [

      "4606d28a-71c7-4c93-8fe4-167a3fd93422"
    ],
    "networkAcls": {
      "bypass": "AzureServices",
      "defaultAction": "Allow",
      "ipRules": [],
      "virtualNetworkRules": []
    },
    "privateEndpointConnections": null,
    "provisioningState": "Succeeded",
    "publicNetworkAccess": "Enabled",
    "regions": [],
    "scheduledPurgeDate": null,
    "securityDomainProperties": {
      "activationStatus": "NotActivated",
      "activationStatusMessage": "Your HSM has been provisioned, but cannot be used
for cryptographic operations until it is activated. To activate the HSM, download
the security domain."
    },
    "softDeleteRetentionInDays": 7,
    "statusMessage": "The Managed HSM is provisioned and ready to use.",
    "tenantId": "10a18477-d533-4ecd-a78d-916dbd849d7c"
  },
  "resourceGroup": "rg-infoarchive-cluster_westus",
  "sku": {
    "family": "B",
    "name": "Standard_B1"
  },
  "systemData": {
    "createdAt": "2025-05-07T09:48:44.989000+00:00",
    "createdBy": "...",
    "createdByType": "User",
    "lastModifiedAt": "2025-05-07T09:48:44.989000+00:00",
    "lastModifiedBy": "...",
    "lastModifiedByType": "User"
  },
  "tags": {},
  "type": "Microsoft.KeyVault/managedHSMs"
}
```

Azure lets you update certain properties of the vault after creation. For example, if you did not enable purge protection, use the following command to enable it:

```
$ az keyvault update-hsm --subscription 59dfc29c-c8dd-4293-823d-e82a93cce062 -g rg-
infoarchive-cluster_westus --hsm-name testhsm --enable-purge-protection true
```

8.1.1 Activating the vault

Activation of the HSM vault requires a number (between 3 and 10) of RSA key-pairs:

```
$ openssl req -newkey rsa:2048 -nodes -keyout cert_0.key -x509 -days 365 -out cert_0.cer
Generating a RSA private key
.....................................+++++
.........+++++
writing new private key to 'cert_0.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:NL
State or Province Name (full name) [Some-State]:Noord-Brabant
Locality Name (eg, city) []:Oosterhout
Organization Name (eg, company) [Internet Widgits Pty Ltd]:.
Organizational Unit Name (eg, section) []:.
Common Name (e.g. server FQDN or YOUR name) []:.
Email Address []:.
```

After creating the desired amount, the following command activates your HSM vault:

```
$ az keyvault security-domain download --hsm-name testhsm --sd-wrapping-keys ./
cert_0.cer ./cert_1.cer ./cert_2.cer --sd-quorum 2 --security-domain-file testhsm-SD.json
{
  "status": "Success",
  "statusDetails": "The resource is active."
}
```

The example command above contained three key-pairs.

8.1.2 Creating a key for encryption and decryption

Use the CLI or portal to add yourself into the role of `Managed HSM Crypto User` for it. For more information, see the section “Step 2: Configure the Managed HSM role assignment” in the Azure document titled *Enable HSM customer-managed keys for managed services*.

After that, you can create the key. Make sure that:

- The key type is RSA. Getting your storage account to use a key of another type will not work.
- Make sure to include `encrypt`, `decrypt`, and `wrapKey` in your enabled key operations. Again, configuring your storage account later with a key without `wrapKey` enabled, will not work

Example 8-2: Example of the command to create a key

```
$ az keyvault key create --hsm-name testhsm --name testhsmkey --ops encrypt decrypt
wrapKey unwrapKey --tags --kty RSA
{
  "attributes": {
    "created": "2025-05-07T15:59:37+00:00",
```

```
    "enabled": true,
    "expires": null,
    "exportable": false,
    "hsmPlatform": null,
    "notBefore": null,
    "recoverableDays": 7,
    "recoveryLevel": "CustomizedRecoverable",
    "updated": "2025-05-07T15:59:37+00:00"
  },
  "key": {
    "crv": null,
    "d": null,
    "dp": null,
    "dq": null,
    "e": "AQAB",
    "k": null,
    "keyOps": [
      "encrypt",
      "decrypt"
    ],
    "kid": "https://testhsm.managedhsm.azure.net/keys/testhsmkey/
2596277b6edc42959473b229d2a9106b",
    "kty": "RSA-HSM",
    "n": "...",
    "p": null,
    "q": null,
    "qi": null,
    "t": null,
    "x": null,
    "y": null
  },
  "managed": null,
  "releasePolicy": null,
  "tags": null
}
```

8.1.3 Configuring the storage account

We mentioned the storage account several times in the previous sections of this chapter. To deploy OpenText Information Archive to store its data into storage encrypted with HSM-backed keys, you need to first configure a storage account to use your HSM vault. After that, storage in the system (for files and/or blobs) needs to be configured to use that storage account.

8.1.3.1 Portal preparation

The following was tested through the portal. To do so, ensure the portal can use the new vault and key for the storage account. Basically, you need to let the vault know that the portal (or rather the Azure Resource Manager) can be trusted. For more information, see the Azure document titled *Allow key management operations through Azure Resource Manager*.

Example 8-3: Example of the command to ensure the portal can use the new vault

```
$ az keyvault setting update --hsm-name testhsm --name
AllowKeyManagementOperationsThroughARM --value true
{
```

```
  "name": "AllowKeyManagementOperationsThroughARM",
  "settingType": "boolean",
  "value": "true"
}
```

8.1.3.2 Managed identity preparation

To further prepare storage account configuration with your new HSM vault. configure a managed identity (create a new or use an existing one) and set it up for the role “Managed HSM Crypto Service Encryption User” on the HSM vault, again through portal (`vault | local rbac`) or CLI.

Once done, you are prepared to configure your storage account to use the HSM vault. In the portal steps for storage account configuration, configure the information in the **Encryption** tab.

8.2 Configuring OpenText Information Archive storage

Once HSM vault is configured in a storage account, you are ready to configure OpenText Information Archive storage to benefit from HSM-backed encryption.

8.2.1 PVCs (Persistent Volume Claim)

Configure a new storage class on your storage account configured to use the HSM vault.

Example 8-4: Example of how to configure a new storage class

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: testhsm-azurefile
provisioner: kubernetes.io/azure-file
allowVolumeExpansion: true
volumeBindingMode: Immediate
mountOptions:
 - dir_mode=0777
 - file_mode=0777
 - uid=1000
 - gid=1000
 - mfsymlinks
 - cache=strict
 - actimeo=30
parameters:
  skuName: Standard_LRS
  storageAccount: sa-testhsm
```

In the example above, the storage account is called `sa-testhsm`. The other parameters/options depend on your own deployment.

This new storage class can now be used in the PVC configuration of your OpenText Information Archive deployment. Note that the system Helm chart includes a `pvcs.yaml` template where the storage class for PVC is made configurable (`.Values.storageClassNameRWM`). Hence, all you need to include in your Helm install command is a values override pointing to a file with the following entry:

```
storageClassNameRWM: testhsm-azurefile
```

8.2.2 Object storage

To get OpenText Information Archive to use object storage from the HSM-backed storage account, configure a new storage system, which can be done via the IA Web App's **Storage** tab or using declarative configuration (DC).

The account name is the name of the storage account (`sa-testhsm` in the previous example). The key can be found through the Azure portal – navigate to your storage account and then to its access keys.

After configuring your new Azure Blob Storage, use the IA Web App to add it to your application's space and start configuring stores on it.

8.3 Migration from old storage account to HSM-backed storage account

It is possible to move an OpenText Information Archive deployment that started out on a storage account without an HSM vault to one with an HSM vault. To accomplish this:

- Migrate your PVC data from one storage account to the other and reconfigure your PVs (Persistent Volume) and PVCs.
- Migrate your object store data from one storage account to the other and reconfigure your storage via the IA Web App.

Tip: Data migration in Azure can be quite slow. We recommend you do some performance testing first on your deployment.

8.3.1 PVC migration

PVC "migration" is the most complex of the two steps. If you do not want to copy your data twice, this is a possible way to proceed:

1. Create a new PVC (with a PV) in your new storage account (for example, using the new storage class that maps to that storage, `testhsm-azurefile` in the previous example).
2. Copy the data from the old PVC's (`pvc-source`) PV (`pv-source`) to the new PVC's (`pvc-target`) PV (`pv-target`). You can use CLI for that with the `azcopy copy` command. However, that requires some patience figuring out the exact parameters and it will also require you to generate "sas tokens" for your storage accounts.

Tip: Instead, you could downloaded Microsoft Azure Storage Explorer, which uses `azcopy` underneath, to do the heavy lifting, reducing the copying of data to a basic select/copy/paste.

3. Edit `pv-target` to have `persistentVolumeReclaimPolicy: Retain` (instead of `Delete`).
4. Delete `pvc-target`, which leaves `pv-target` alone in a `Released` state.
5. Edit `pv-target` again to remove its `claimRef:` entry. this leaves `pv-target` in a state where it can be reclaimed by another PVC.
6. Delete `pvc-source`. You can set `pv-source`'s `persistentVolumeReclaimPolicy` to `Retain` if you want to keep the old data just in case.
7. Now, re-create `pvc-source` with the same name, but with the following changes:
   a. `storageClassName` set to the new storage class (`testhsm-azurefile` in the previous example).
   b. `volumeName` set to the name of `pv-target` .

   Now you have your PVC pointing to the new PV.
8. Edit `pv-target` again to set `persistentVolumeReclaimPolicy` back to `Delete`.

8.3.2 Blob container migration

For the migration of object storage:

1. Create a new container in the new storage account with the same name as the old container.
2. Copy the data from the old to the new container. Again, you can use CLI for this or Microsoft Azure Storage Explorer.
3. Use the IA Web App to edit the Azure storage, changing the **Account Name** and **Account Key** fields to match those of your new (HSM-backed) storage account.

Chapter 9

Appendix

9.1 A: Interview questions asked when encrypting passwords and secrets

For additional information about the parameters, see Configuring crypto objects.

`Use Vault to store configuration secrets`

- If you are using Vault, type `y`; otherwise, type `n`.

`Password encryption security provider {1=SUN_JCE, 2=BOUNCY_CASTLE, 3=GEMALTO}:`

`Password encryption keystore type {1=PKCS12, 2=JCEKS}:`

`Password encryption keystore:`

`Password encryption key ID:`

`Password encryption algorithm {1=AES}:`

`Password encryption keySize {1=128, 2=192, 3=256}:`

`Password encryption mode {1=CBC, 2=CFB, 3=CTR, 4=CTS, 5=OFB, 6=PCBC}:`

`Password encryption padding scheme {1=NOPADDING, 2=PKCS5PADDING, 3=ISO10126PADDING}:`

`Enable no prompt startup: (y/N):`

- When passwords are encrypted, each component needs the master password to enable decryption of passwords. Normally this password will be asked interactively on the console when starting each component. However in order to automate service startup it is necessary to provide this password through different means. Enabling the no prompt startup stores the master password in a secure manner on local disk so no manual interaction is required during startup.

`Enable use of Gemalto for data encryption`:

- If you are using Gemalto (now called CipherTrust), type `y`; otherwise, type `n`.
- Type `y`.

9.1.1 Gemalto

These questions are asked if you selected Gemalto as the password encryption security provider. To learn more about configuring Gemalto, see Connecting system Components to CipherTrust/KeySecure Appliance.

`Gemalto user name:`

`Gemalto client certificate name:`

`Gemalto propertiesFile: (<IngrianNAE.properties>):`

- This is the property file that you configured (the `IngrianNAE.properties` file).

`Gemalto group`

`Password encryption keystore password:`