Properties Encryption

Business Value to Client

By default, OpenLegacy projects contain various passwords, credentials, and tokens required to connect to backend systems, databases or other system components. To avoid sensitive data leakage, we should avoid saving it as clear-text across projects and ecosystems.

Potentially Impacted Components in the Product

- OpenLegacy Projects of all types: SDK, API and Microservices
- OpenLegacy Microservices Ecosystem (especially OpenLegacy Config Server)
- OpenLegacy IDE (Project Creation Wizards)
- OpenLegacy MMC

Prerequisites

Properties Encryption functionality relies on **Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files.**

For **JDK** version lower than **1.8.0_u161,** this extension must be installed to enable Properties Encryption feature.

For **OpenJDK8u151** and higher, please refer to the following documentation https://github.com/open-eid/cdoc4j/wiki/Enabling-Unlimited-Strength-Jurisdiction-Policy.

Starting with **JDK** version **1.8.0_u161**, **JCE** files are included in the **JDK** distribution. Therefore, no additional actions are required if you are using newer versions of the JDK.

> **Note:** If Unlimited Strength Policy is not provided on the host **JDK**, Properties Encryption feature will be **disabled** and sensitive data will be saved as clear-text.

The OpenLegacy Installation comes with **jdk1.8.0_u201** out of the box, and the Properties Encryption feature is enabled by default.

Properties Encryption Feature

The ultimate goal of this feature is to enable the encryption of sensitive data stored in the application properties files. Once encrypted, it should appear in the property file as ciphertext. Original data cannot be recovered without having an encryption key. This feature prevents malicious use of this data, should project source code is leaked or passwords are exposed in another way.

Given that **JCE Unlimited Strength Policy** is provided and Encryption feature is not disabled by setting **`encrypt.disabled=true`**, the following stages would take place during OpenLegacy application creation and startup:

Design-time

1. If **OpenLegacy Project Creation Wizard** contains password fields, the mechanism of **Encryptor Creation** is engaged
2. **Properties Encryption Service** loads **Encryption Properties** in several locations (**Encryption Properties Loader Stage 1**)
3. **Encryptor** is created based on **Encryption Properties** loaded in the previous stage
4. Clear-text password is converted to Ciphertext using the **Encryptor** which was created previously.
5. Passwords in ciphertext is saved to project's ***.yml** configuration files.

Once the project is created, you can add additional properties. If you want to encrypt any sensitive properties you should do it manually by using Spring CLI.

> **Note:** Encrypted properties have a strict format in which they should appear in the YAML file. A key should be followed by a string value surrounded by single-quotes and starting with "{cipher}" prefix (without quotes).
> For example:
> `password: '{cipher}bd046712b4144506742ed2815272f875b48705659bac4ce1c7b72fdd739e937a'`

Run-time

1. **Properties Encryption Service** loads **Encryption Properties** in several locations (**Encryption Properties Loader Stage 2**)
2. **Encryptor** is created based on **Encryption Properties** loaded in previous stage
3. Ciphertext properties are converted to clear-text using the **Encryptor** which was created previously and saved in application memory.

Text Encryptor mentioned before is an implementation of spring interface **`org.springframework.security.crypto.encrypt.TextEncryptor`** which uses the **AES/CBC** encryption algorithm by default.

Configuration

To enable/disable or configure Properties Encryption behavior, you can utilize the following properties:

Property	Default Value	Type	Description	Environment Variable
encrypt.disabled	*false*	Boolean	Property used to disable encryption	**OL_DISABLE_ENCRYPTION**
encrypt.key	*changeme*	String	A symmetric encryption key. Consider using a keystore as a stronger alternative.	**OL_ENCRYPTION_KEY**
encrypt.salt	*deadbeef*	String (HEX)	Salt for the random secret used to encrypt ciphertext. Once it is set, do not change it (or existing ciphers will not be decryptable).	**OL_ENCRYPTION_SALT**
encrypt.fail-on-error	*true*	Boolean	Flag raised if a process fails due to an encryption or decryption error.	**OL_ENCRYPTION_FAIL_ON_ERROR**
encrypt.key-store.location	*null*	String	Location of the keystore file, e.g. classpath:keystore.jks, file:C:/Users/Admin/keystore.jks (Internally loaded as org.springframework.core.io.Resource using org.springframework.core.io.DefaultResourceLoader)	**OL_KEYSTORE_LOCATION**
encrypt.key-store.password	null	String	Password that locks the keystore.	**OL_KEYSTORE_PASSWORD**
encrypt.key-store.alias	null	String	Alias for a key in the store.	**OL_KEYSTORE_ALIAS**
encrypt.key-store.secret	null	String	Secret protecting the key (defaults to the same as the password).	**OL_KEYSTORE_SECRET**
encrypt.rsa.algorithm	*DEFAULT*	Enum	The RSA algorithm to use (DEFAULT or OEAP). Once it is set, do not change it (or existing ciphers will not be decryptable).	**OL_ENCRYPTION_RSA_ALGORITHM**
encrypt.rsa.strong	false	Boolean	Flag to indicate that "strong" AES encryption should be used internally. If true, the GCM algorithm is applied to the AES encrypted bytes. Default is false (in which case, "standard" CBC is used). Once it is set, do not change it (or existing ciphers will not be decryptable).	**OL_ENCRYPTION_RSA_STRONG**
encrypt.rsa.salt	deadbeef	String (HEX)	Salt for the random secret used to encrypt cipher text. Once it is set, do not change it (or existing ciphers will not be decryptable).	**OL_ENCRYPTION_RSA_SALT**

These properties can be defined in one of the following locations:

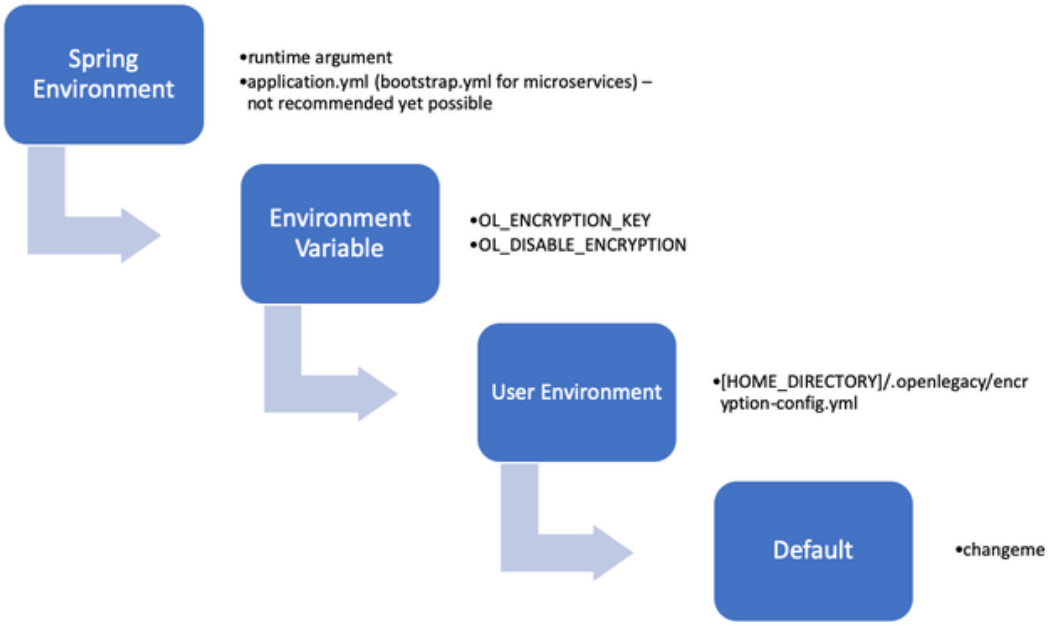
1. Spring environment - via program arguments or properties file. **application.yml** for regular project and **bootstrap.yml** for microservices project.
2. System environment - via Environment Variable
3. User environment - via '[HOME_DIRECTORY]/.openlegacy/encryption-config.yml' file

If there are no properties in these locations, default properties are applied

> While it is possible to define encryption properties in application configuration files (application.yml, bootstrap.yml), it is highly recommended not to use this option except in testing.

Encryption Properties Loading Priority

The following priority is applied if encryption properties are defined in multiple sources (from higher to lower):



Encryption Property Source	Design-time*	Run-time	Microservices	Config Server
Spring Environment	There is no way to initialize Spring environment before project is created	application.yml or command-line argument	bootstrap.yml or command-line argument	bootstrap.yml or command-line argument
System Environment (Env. Variable)				
User Environment				
Default				

*** Design-time** - in this context refers to Project Creation stage (Project Creation Wizards)

Microservice Ecosystem and MMC

Passwords and tokens in ecosystem projects and MMC are encrypted with the default key (**changeme**)**.** Currently, there is no way to automatically encrypt sensitive data upon ecosystem import, as ecosystem projects and MMC are imported from zip files and are not generated by Freemarker.

If you are going to use a different encryption key, you will have to replace all the ciphers manually.

Config Server

Spring Config Server supports Properties Encryption out of the box. The OpenLegacy Properties Encryption module is responsible for creation of the TextEncryptor.

Additionally, the OpenLegacy Config Server provides proprietary functionality to encrypt sensitive data upon saving or updating configuration files.

You can manage a list of default property-keys to encrypt by providing the following property n the YAML file:

Property	Default Value	Type	Description
ol.environment.keys-to-sanitize		**List**	Keys defined by this property will be encrypted before persisting to database or git

Manual Encryption

Property encryption and re-encryption during all stages of development cannot be fully automated. Once an OpenLegacy project is created, a

user must encrypt additional properties and re-encrypt properties if an encryption key is changed.

For this purpose Spring CLI should be used, coupled with the Spring Cloud extension.

> **Note:** the Spring CLI version and Spring Boot version used by OpenLegacy must match. As of writing this article, the current OpenLegacy Spring Boot version was 1.5.19.RELEASE.

Common Pitfalls

1. Once an encryption key is configured, don't change it unless you change all the ciphers as well. This will lead to an error during application startup.
2. Don't change any properties except *encrypt.key* and *encrypt.disabled* unless necessary. Be extremely careful when changing properties, as errors in this could have severe consequences on your security.
3. Upon disabling the Encryption feature, make sure you get rid of all ciphers in configuration properties. Otherwise, the application will fail on startup.

Relevant Tickets

OL-5762

Implementation Notes, Relevant Docs and Examples