|  |  |
| --- | --- |
| February 18, 2011 |  |
|  | Security Framework |
| Version 1.0 |  |
| Encryptor user's guide | |



Table of Contents

1 Introduction 1

2 Configuration file structure 1

2.1 EncryptorProperties 1

2.2 Engine 2

2.2.1 Non-configurable engines 2

2.2.2 Configurable engines 3

3 Encryptor subsystem API 4

# Introduction

Encryptor subsystem provides tool to protect sensitive data such as passwords, credit card properties, etc.

It includes encrypting/decrypting the data with RSA and AES algorithms, hashing with RSA and SHA, utilizing MS DPAPI.

Subsystem provides an API to be used by a client application.

# Configuration file structure

Configuration is a set of XML documents. Each Security Framework subsystem has a separate configuration file. There is a root configuration file - MtSfConfigurationLoader.xml that references all subsystem’s configuration files.

Please see the section “2.1. Security Monitor configuration reference” in the Security Monitor user’s guide for the root configuration file reference.

The Encryptor configuration file structure at high level is shown below:

|  |
| --- |
| <EncryptorProperties IsRuntimeApiEnabled="true" IsRuntimeApiPublic="true" IsControlApiEnabled="true" IsControlApiPublic="true">  <PasswordConfig />  <CryptoConfig />  <RsaCryptoConfig />  <SessionKeyConfig />  <Engines>  <Engine />  <Engine />  </Engines>  </SubsystemProperties> |

The following sections describe attributes, elements, and child elements.

## EncryptorProperties

Root element. It’s not required but recommended do not change the element’s name. It matches with name of the class that takes the configuration properties when it is loading.

There are the following child elements and attributes under the root.

Attributes:

| Attribute | Description |
| --- | --- |
| IsRuntimeApiEnabled | Required attribute.  Indicates whether the Encryptor API is enabled. Prevents the API from initializing if set to **false**. |
| IsRuntimeApiPublic | Required attribute.  Indicates whether the Encryptor API is exposed to public access. |
| IsControlApiEnabled | Required attribute.  Indicates whether the Encryptor control API is enabled. Prevents the control API from initializing if set to **false**. *For future use*. |
| IsControlApiPublic | Required attribute.  Indicates whether the Encryptor control API is exposed to public access. *For future use*. |

Elements:

| Element | Description |
| --- | --- |
| PasswordConfig | Required element.  Reserved for future use. |
| CryptoConfig | Required element.  Reserved for future use. |
| RsaCryptoConfig | Required element.  Reserved for future use. |
| SessionKeyConfig | Required element.  Reserved for future use. |
| Engines | Required element.  Defines available engines and their properties. |

The Engine element’s structure is described below.

## Engine

### Non-configurable engines

There are 2 engines those cannot be configured:

1. ProtectDataEngine – provides data encryption using Windows DPAPI.
2. UnprotectDataEngine – provides decryption for the data protected with Windows DPAPI.

The following attributes are available for them in the Engine element:

<SubsystemProperties>

<Engines>

|  |
| --- |
| <Engine Id="<Engine\_Id>"IsDefault="true|false" RealType="<Assembly\_qualified\_type\_name>" /> |

| Attribute | Description |
| --- | --- |
| Id | Required attribute.  A unique ID of the engine. The engine can be gotten by this value via Encryptor API. |
| IsDefault | Optional attribute.  Specifies whether the engine is default for its category.  Value: Boolean (default False). |
| RealType | Required attribute.  Defines a class taking the engine’s configuration.  Value: the following engines are defined in the subsystem   * MetraTech.SecurityFramework.Core.Encryptor.ProtectDataEngine, MetraTech.SecurityFramework * MetraTech.SecurityFramework.Core.Encryptor.UnprotectDataEngine, MetraTech.SecurityFramework |

### Configurable engines

The following engines can be configured:

1. MSHashEngine – provides hashing with HMAC-SHA-512 algorithm
2. MSEncryptEngine – provides encryption with symmetric AES algorithm
3. MSDecryptEngine – provides decryption of the data encrypted with symmetric AES algorithm
4. RSAHashEngine – provides hashing with HMAC‐SHA‐256 algorithm
5. RSAEncryptEngine – provides encryption with RSA algorithm
6. RSADecryptEngine – provides decryption of the data encrypted with RSA algorithm

The following attributes are available for them in the Engine element:

<SubsystemProperties>

<Engines>

|  |
| --- |
| <Engine Id="<Engine\_Id>"IsDefault="true|false" RealType="<Assembly\_qualified\_type\_name>" KeyClassName="<Engine\_specific>" /> |

| Attribute | Description |
| --- | --- |
| Id | Required attribute.  A unique ID of the engine. The engine can be gotten by this value via Encryptor API. |
| IsDefault | Optional attribute.  Specifies whether the engine is default for its category.  Value: Boolean (default False). |
| RealType | Required attribute.  Defines a class taking the engine’s configuration.  Value: the following engines are defined in the subsystem   * MetraTech.SecurityFramework.Core.Encryptor.MSHashEngine, MetraTech.SecurityFramework * MetraTech.SecurityFramework.Core.Encryptor.MSEncryptEngine, MetraTech.SecurityFramework * MetraTech.SecurityFramework.Core.Encryptor.MSDecryptEngine, MetraTech.SecurityFramework * MetraTech.SecurityFramework.Core.Encryptor.RSAHashEngine, MetraTech.SecurityFramework * MetraTech.SecurityFramework.Core.Encryptor.RSAEncryptEngine, MetraTech.SecurityFramework * MetraTech.SecurityFramework.Core.Encryptor.RSADecryptEngine, MetraTech.SecurityFramework |
| KeyClassName | Required attribute.  Defines a name of the encryption key to be used.  Value: the following values are available for different engine types   * MSHashEngine – PasswordHash, PaymentMethodHash * MSEncryptEngine – PaymentInstrument, DatabasePassword, ServiceDefProp, Ticketing, QueryString * MSDecryptEngine – PaymentInstrument, DatabasePassword, ServiceDefProp, Ticketing, QueryString * RSAHashEngine – PasswordHash, PaymentMethodHash * RSAEncryptEngine – PaymentInstrument, DatabasePassword, ServiceDefProp, QueryString * RSADecryptEngine – PaymentInstrument, DatabasePassword, ServiceDefProp, QueryString |

# Encryptor subsystem API

Simplified Encryptor subsystem API is provided by MetraTech.SecurityFramework.EncryptorExtensions class and consists of extension methods for the System.String class.

To use the simplified API the reference to SecurityFramework.dll must be added and MetraTech.SecurityFramework must be declared in using directive.

The following methods are defined in the API:

| Method | Description |
| --- | --- |
| ProtectData(input: string) : string | Invokes a default Protect data engine passing the string instance as an input.  Returns an encryption result (as a BASE 64 encoded string). |
| UnprotectData(input: string) : string | Invokes a default Unprotect data engine passing the string instance as an input.  Returns a decryption result (as a string).  Throws the SubsystemInputParamException when some invalid data was passed to the method. |
| EncryptAes(keyClassName: string): string | Encrypts the data with symmetric AES algorithm using a key specified by the keyClassName argument.  Returns an encryption result (as a BASE 64 encoded string).  Throws the SecurityFrameworkException when the specified key not found. |
| DecryptAes(keyClassName: string): string | Decrypts the data encrypted with symmetric AES algorithm using a key specified by the keyClassName argument.  Returns an decryption result (as a string).  Throws the SecurityFrameworkException when the specified key not found.  Throws the SubsystemInputParamException when some invalid data was passed to the method. |
| HashSha(keyClassName: string, keyId: string): string | Generates a SHA-512 hash from the string instance using the specified key (by its ID).  Returns a hash (as a BASE 64 encoded string) prepended with an ID of the used key.  Throws the SubsystemInputParamException when the specified key not found. |
| HashSha(keyClassName: string): string | Generates a SHA-512 hash from the string instance using the specified key (by its name).  Returns a hash (as a BASE 64 encoded string) prepended with an ID of the used key.  Throws the SubsystemInputParamException when the specified key not found. |
| CompareWithHashSha(keyClassName: string, hash: string): bool | Compares a SHA-512 hash obtained from the string instance using the specified key with the specified value.  Returns True if the hash exactly matches the specified value (hash argument) and false otherwise.  Throws the SubsystemInputParamException when the specified key not found. |
| ParseKeyFromHash(): string | Extracts a key ID from the output of the HashSha method.  Throws the FormatException if the string instance does not start with a valid GUID. |
| CreateSessionKeys(keyClassNames: IEnumerable<string>): IDictionary<string, string> | Create one session key for each key class. If the given string instance is empty, a hardcoded password is used. Existing keys, if any, are deleted. Output is generated in RMP\config\security\sessionkeys.xml.  A dictionary containing keys and passwords to them.  Throws the SecurityFrameworkException if some of the passed key names is invaid. |
| CreateCryptographicKey(string keyClassName) | Create a session key for the specified key class based on the string instance as the password. The existing keys for the specified keyclass will be deleted.  Throws the EncryptorInputDataException if the specified key name is invalid. |
| CreateCryptographicKey(base64Key: out string, base64Iv: out string) | Creates a random key and initialization vector for symmetric AES based on the string instance as a password. |
| AddCryptographicKey(keyClassName: string, id: Guid, makeCurrent: bool) | Create a session key for the specified key class based on the specified password and identifier. If makeCurrent is true, the key will be made the current key for the given key class.  Throws the EncryptorInputDataException if the specified key name is invalid.  Throws the EncryptorInputDataException if the specified key ID is invalid. |
| EncryptRsa(keyClassName: string): string | Encrypts the data with RSA algorithm using a key specified by the keyClassName argument.  Returns an encryption result (as a BASE 64 encoded string).  Throws the SecurityFrameworkException when the specified key not found. |
| DecryptRsa(keyClassName: string): string | Decrypts the data encrypted with RSA algorithm using a key specified by the keyClassName argument.  Returns an decryption result (as a string).  Throws the SecurityFrameworkException when the specified key not found.  Throws the SubsystemInputParamException when some invalid data was passed to the method. |
| HashRsa(keyClassName: string, string keyId): string | Generates a SHA-256 hash from the string instance using the specified key (by its ID).  Returns a hash (as a BASE 64 encoded string) prepended with an ID of the used key.  Throws the SubsystemInputParamException when the specified key not found. |
| CompareHashRsa(keyClassName: string, hash: string): bool | Compares a SHA-256 hash obtained from the string instance using the specified key with the specified value.  Returns True if the hash exactly matches the specified value (hash argument) and false otherwise.  Throws the SubsystemInputParamException when the specified key not found. |
| ParseKeyFromHashRsa(): string | Extracts a key ID from the output of the HashRsa method. |