Solutions to Protect the OpenPGP Secret Passphrase

setup Guide

Jonathan Keng Hung Aw

2023

Contents

[Introduction 2](#_Toc135994319)

[Key Prerequisites to run GnuPG on your Development Machine/Server: 2](#_Toc135994320)

[**1.** The Importance of a Passphrase 3](#_Toc135994321)

[2. Setup Steps for Solution 1: Protect Passphrase using AWS Secrets Manager 4](#_Toc135994322)

[3. Setup Steps for Solution 2: Protect Passphrase using Windows Data Protection API 11](#_Toc135994323)

[4. Setup Steps for Solution 3: Protect Passphrase using ASP.NET Core Data Protection API 15](#_Toc135994324)

# Introduction

This document provides the instructions to setup the solutions in MOH OpenPGP Implementation Guide’s **Developer Starter Kit** to protect your OpenPGP Secret Passphrases on your Development Machine/Server.

Refer to this document to prepare the **OpenPGP Passphrase protection** configurations needed to execute the runtime of the following projects in the starter kit:

1. **OpenPgpBatchJob**
2. **PgpCombinedCrypto**

Target audience for this document is:

1. Someone who has read the [SETUP.docx](https://github.com/jawkh/gpgme-sharp-implementation-guide/tree/master/OpenPgpBatchJob/SETUP.docx) for the OpenPgpBatchJob.
2. A developer who is familiar with how to clone the github repository and use Visual Studio to configure, build and run the source code in the developer starter kit.

# Key Prerequisites to run GnuPG on your Development Machine/Server:

**Install GnuPG on your Development Machine. This is a key prerequisite to run GnuPG (i.e., which provides the OpenPGP capabilities) on your Computer/Server.**

|  |
| --- |
| * On Windows, you will need to install [Gpg4Win](https://www.gpg4win.org/). Note that Gpg4Win currently only distributes a 32-bit build, so on Windows you **must** set your C# app to run in 32-bit mode. * On Debian and Ubuntu, install the [libgpgme11 package](https://packages.debian.org/stretch/libgpgme11). * On other Linux distros or other operating systems, install libgpgme using your favourite package manager, or compile it from source. |

For security reason, it is extremely important to protect the confidentiality of the Secret Passphrases of the OpenPGP Private Keys.

|  |
| --- |
| The Importance of a Passphrase A passphrase generally refers to a secret used to protect an encryption key.  The purpose of the passphrase is usually to encrypt the private key. This makes the key file by itself useless to an attacker. To use an encrypted key, the passphrase is also needed. In a way, they are two separate factors of authentication.  In the case of GnuPG, the passphrase is required to perform OpenPGP operations involving an OpenPGP private key, eg. decryption and signing. More importantly, the passphrase is required for exporting the private keys from the GnuPG keystore on your server.  Therefore, it is extremely important to protect the confidentiality of the passphrases of your OpenPGP private keys, lest a malicious insider who has access to your server can export your OpenPGP private keys and use them elsewhere without consent. Therefore, the Security Hardening and Security Coding Standards place emphasis on protecting the confidentiality of the passphrases. |

Consequently, this **Developer Starter Kit** provides 3 ready-to-use solutions that protects the confidentiality of the secret passphrases of your OpenPGP private keys.

1. Solution 1: Uses **AWS Secrets Manager** [Recommended for AWS serverless and containerized based solutions. Also useful for Applications hosted on AWS EC2 Instances.]
2. Solution 2: Uses **Windows Data Protection API** [Only works for Systems developed for Windows OS. Optimized for Windows-Based Applications!]
3. Solution 3: Uses **ASP.NET Core Data Protection API** [Works for Windows, Linux and macOS based Applications. Can be used on any .NET core applications, including non-ASP.NET ones. Recommended for all other types of Applications that cannot use Solutions 1 & 2.]

Firstly, decide on the method for protecting the confidentiality of the OpenPGP Private Keys’ secret passphrases on your Development Machine/Server.

Then follow the Setup Steps provided in this document for the chosen option.

# Setup Steps for Solution 1: Protect Passphrase using AWS Secrets Manager

1. Login to AWS Console and go to AWS Secrets Manager Console

<https://ap-southeast-1.console.aws.amazon.com/secretsmanager/newsecret?region=ap-southeast-1>

Graphical user interface, application

Description automatically generated

Figure : Choose to create a new secret. Select **Other type of secret**. Configure a **Key/Value** pair to store the secret passphrase itself.

Graphical user interface, text, application, email

Description automatically generated

Figure : Assign a **Secret name** and Description. The **Secret name** will be used to retrieve the secret programmatically.

Graphical user interface, application

Description automatically generated

Figure : Assign a **AccessProject** Tag for this secret. This is used to implement additional project-specific access-control for this Secret in the IAM Resource policy.

Graphical user interface, text, application, email

Description automatically generated

Figure : Assign IAM Resource Policy for this Secret. (example)

|  |
| --- |
| {  "Version": "2012-10-17",  "Statement": {  "Effect": "Allow",  "Principal": {  "AWS": [ARN for IAM User(s)/Role(s) that shall be granted permission to access to this secret]  },  "Condition": {  "StringEquals": {  "aws:ResourceTag/AccessProject": "${ aws:PrincipalTag/AccessProject }"  }  },  "Action": ["secretsmanager:GetSecretValue","secretsmanager:DescribeSecret"],  "Resource": "\*"  }  } |

Figure : Sample IAM Resource Policy to be assigned to this Secret.

1. Go to AWS IAM console.

<https://us-east-1.console.aws.amazon.com/iamv2/home?region=ap-southeast-1#/home>

Graphical user interface, application

Description automatically generated

Figure : Ensure you assign the identical value for the **AccessProject** tag to the target IAM User/Role that was granted the permission to access the Secret in the IAM Resource Policy that was configured in the preceding step.

1. Setup **AWS Toolkit for Visual Studio** on your Development Machine. With this, you can securely store your AWS IAM user credentials on your development machine and need not hardcode the sensitive credentials into your source code or config file.

References:

* <https://docs.aws.amazon.com/toolkit-for-visual-studio/latest/user-guide/credentials.html>
* <https://docs.aws.amazon.com/toolkit-for-visual-studio/latest/user-guide/keys-profiles-credentials.html>
* <https://docs.aws.amazon.com/toolkit-for-visual-studio/latest/user-guide/basic-use.html>

A screenshot of a computer

Description automatically generated

Figure : Configuring AWS IAM Account Profile using AWS Toolkit for Visual Studio.NET

Text

Description automatically generated

Figure : Select the AWS IAM profile to use on your development machine.

|  |
| --- |
| **Important – setup for hosting environments:**  Use one of the following options for supplying AWS Credentials in an **AWS Hosting Environment**.   * Amazon EC2 Instance Metadata Service ([IAM role attached to an instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html))   The SDK attempts to fetch credentials from the [Instance Metadata Service](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html).   * Amazon ECS container credentials ([IAM roles for task](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-iam-roles.html))   The SDK attempts to resolve AWS\_CONTAINER\_CREDENTIALS\_RELATIVE\_URI or AWS\_CONTAINER\_CREDENTIALS\_FULL\_URI environment variables to fetch credentials from.   * AWS STS web identity (including Amazon Elastic Kubernetes Service (Amazon EKS))   The SDK attempts to resolve JVM system properties and environment variables to [assume a role using a web identity](https://docs.aws.amazon.com/STS/latest/APIReference/API_AssumeRoleWithWebIdentity.html). |

1. Configure the appSettings in app.config file of the **PgpCombinedCrypto** project.
   * Set the value for **SenderDataFilePath**. You can use this value to supply your own data files for testing. Otherwise, you could just use the default value.
   * Set the value for **PassphraseProtectionMode** to AWS\_SECRETSMANAGER
   * Set value for **AliceEmailAddress.** Enter the email of the sender’s OpenPGP keypair.
   * Set value for **AliceAWSSecretsName.** Enter the Secret Name of the AWS Secrets Manager Secret that stores the sender’s OpenPGP Private Key Secret Passphrase OpenPGP.
   * Set value for **BobEmailAddress** Enter the email of the recipient’s OpenPGP keypair.
   * Set value for **BobAWSSecretsName.** Enter the Secret Name of the AWS Secrets Manager Secret that stores the recipient’s OpenPGP Private Key Secret Passphrase OpenPGP.

Note: The other appSettings parameters that aren’t mentioned above are irrelevant for this option.

**Tip:** You can still change the values in the app.config after building the application by accessing the **PgpCombinedCrypto.dll.config** file in the application folder.

Setup is complete!

Text

Description automatically generated

Figure : Sample Screenshot of a successful execution of **PgpCombinedCrypto.exe**.

# Setup Steps for Solution 2: Protect Passphrase using Windows Data Protection API

**Encrypting the Secret Passphrase.**

1. Configure the appSettings variables in app.config file of the **ProtectSecretsWithWindowsDataProtectionAPI** project.
   * Set the value for **entropy.** Enter a unique value for your application. This acts as a unique identifier (i.e. secondary entropy) for Windows Data Protection API (DPAPI) to perform encryption/decryption operations specifically for your application. Ensure that you specify the same values for the **ProtectSecretsWithWindowsDataProtectionAPI** Console Program, **PgpCombinedCrypto** and your own Application (i.e. the one that implement gpgme-sharp).

Text

Description automatically generated

**Tip:** You can still change the values in the app.config after building the application by accessing the **ProtectSecretsWithWindowsDataProtectionAPI.dll.config** file in the application folder.

1. Build the **ProtectSecretsWithWindowsDataProtectionAPI** project (SHIFT + F6). You should be able to build this project successfully without any errors.

Then, either run the **ProtectSecretsWithWindowsDataProtectionAPI.exe** Console Program directly in the application folder building the project, OR

Run the project directly from Visual Studio.

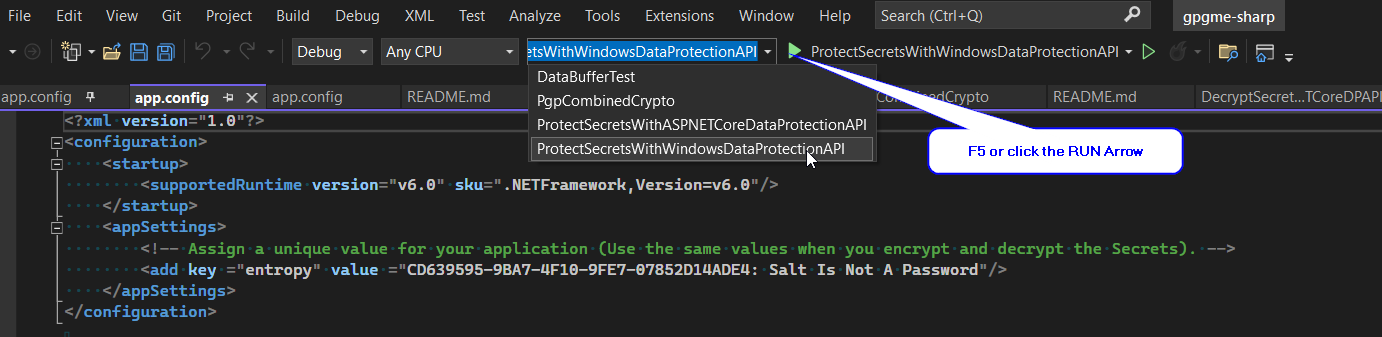


Figure : Run **ProtectSecretsWithWindowsDataProtectionAPI** project from Visual Studio

1. Use the **ProtectSecretsWithWindowsDataProtectionAPI** Console Program to encrypt the secret Passphrases of the OpenPGP Private Keys.

Graphical user interface, text, application

Description automatically generated

Figure : Encrypt the Secret Passphrase using the **ProtectSecretsWithWindowsDataProtectionAPI** Console Program. Copy the encrypted passphrase for use later.

|  |
| --- |
| **Important – setup for Hosting Environment:**  The Windows Data Protection API (DPAPI) is focused on providing data protection for each windows user accounts. This means that the encryption and decryption operations must be done using the same windows account.  <https://learn.microsoft.com/en-us/previous-versions/ms995355(v=msdn.10)?redirectedfrom=MSDN>  Hence, it is imperative to run the **ProtectSecretsWithWindowsDataProtectionAPI** Console Program (to encrypt the secret passphrases) individually on each server, using the same service-account that will be used to run your own Application (i.e. the one that implements gpgme-sharp). Also ensure that you set the same value for **entropy** AppSettings variable for the **ProtectSecretsWithWindowsDataProtectionAPI** console program and for your own Application.  The encrypted passphrase can only be decrypted on the same server where it was originally encrypted.  Copy the **ProtectSecretsWithWindowsDataProtectionAPI** Console Program (i.e. the entire application folder) and run it on each server that needs to perform PGP operations.    Figure : Open File Explorer and browse to the executable file you wish to run as different user. Simply hold down the Shift key and right-click on the executable file, select Run as different user from the context menu. |

1. Configure the appSettings variables in app.config file of the **PgpCombinedCrypto** project.
   * Set the value for **SenderDataFilePath**. You can use this value to supply your own data files for testing. Otherwise, you could just use the default value.
   * Set the value for **PassphraseProtectionMode** to WINDOWS\_DPAPI
   * Set the value for **entropy.** Enter a unique value for your application. This acts as a unique identifier (i.e. secondary entropy) for Windows Data Protection API (DPAPI) to perform encryption/decryption operations specifically for your application. Ensure you specify the same values for the **ProtectSecretsWithWindowsDataProtectionAPI** Console Program, **PgpCombinedCrypto** and your own Application (i.e. the one that implement gpgme-sharp).
   * Set the value for **AliceEmailAddress.** Enter the email of the sender’s OpenPGP keypair.
   * Set the value for **AliceEncryptedSecretPassPhrase\_WIND\_DPAPI.** Enter the encryptedpassphrase generated on the same machine in the preceding step.
   * Set the value for **BobEmailAddress.** Enter the email of the recipient’s OpenPGP keypair.
   * Set the value for **BobEncryptedSecretPassPhrase\_WIND\_DPAPI.** Enter the encrypted passphrase generated on the same machine in the preceding step.

Note: The other appSettings parameters that aren’t mentioned above are irrelevant for this option.

**Tip:** You can still change the values in the app.config after building the application by accessing the **PgpCombinedCrypto.dll.config** file in the application folder.

Setup is complete!

Text

Description automatically generated

Figure : Sample Screenshot of a successful execution of PgpCombinedCrypto.exe

# Setup Steps for Solution 3: Protect Passphrase using ASP.NET Core Data Protection API

**Encrypting the Secret Passphrase.**

1. Configure the appSettings variables in app.config file of the **ProtectSecretsWithASPNETCoreDataProtectionAPI** project.
   * Set the value for **entropy.** Enter a unique value for your application. Ensure that you specify the same values for the **ProtectSecretsWithASPNETCoreDataProtectionAPI** Console Program, **PgpCombinedCrypto** and your own Application (i.e. the one that implement gpgme-sharp).
   * Set the value for **SSLCertDistinguishedSubjectName.** Enter a unique value for your application. This is for generating a self-sign SSL Cert that will be automatically created and installed on your development machine/server when you first run this console program on your machine to perform an encryption of a secret passphrase. This SSL Cert will be used to encrypt and decrypt the ASP.NET Core Data Protection Masterkey in your keyring. Ensure you specify the same values for the **ProtectSecretsWithASPNETCoreDataProtectionAPI** Console Program, **PgpCombinedCrypto** and your own Application (i.e. the one that implement gpgme-sharp).A screenshot of a computer

     Description automatically generated with medium confidence

**Tip:** You can still change the values in the app.config after building the application by accessing the **ProtectSecretsWithASPNETCoreDataProtectionAPI.dll.config** file in the application folder.

1. Build the **ProtectSecretsWithASPNETCoreDataProtectionAPI** project (SHIFT+F6). You should be able to build this project successfully without any errors. This project provides a console program for encrypting your secret passphrase.

Either run the **ProtectSecretsWithASPNETCoreDataProtectionAPI.exe** Console Program directly in the application folder after successfully building this project, OR

Run the project directly from Visual Studio.

A screenshot of a computer

Description automatically generated with medium confidence

Figure : Run **ProtectSecretsWithASPNETCoreDataProtectionAPI** project from Visual Studio

1. Use the **ProtectSecretsWithASPNETCoreDataProtectionAPI** Console Program to encrypt the secret Passphrases of the OpenPGP Private Keys.

Graphical user interface, text, application, chat or text message, Teams

Description automatically generated

Figure : Encrypt the Secret Passphrase using the **ProtectSecretsWithASPNETCoreDataProtectionAPI** Console Program. Copy the encrypted passphrase into the app.config.

|  |
| --- |
| **Important – setup for Hosting Environment:**  The ASP.NET Core Data Protection API provides data protection for apps running on a single machine. ASP.NET Core Data Protection uses its data protection Master key to encrypt, and decrypt protected data, but it doesn’t protect the key itself. On Windows, ASP.NET Core Data Protection encrypts the key using DPAPI. Since DPAPI isn’t available on Linux and macOS, the key is unencrypted and stored as plaintext. If a hacker or another user steals the key, they would be able to decrypt the application data.  Fortunately, ASP.NET Core Data Protection provides developers with multiple ways to encrypt the keys at rest.  The **ProtectSecretsWithASPNETCoreDataProtectionAPI** console program and code library useX.509 certificate to encrypt the ASP.NET Core Data Protection Master key at rest. This program will auto-generate a Self-Signed SSL Cert and import it into the **Current User’s Personal Cert Store** for this purpose.  Hence, it is imperative to run the **ProtectSecretsWithASPNETCoreDataProtectionAPI** Console Program (to encrypt the secret passphrases) on each server, using the same service-account that will be used to run your own Application (i.e. the one that implements gpgme-sharp). Also ensure that you set the same value for **entropy** and **SSLCertDistinguishedSubjectName** AppSettings variables for the **ProtectSecretsWithASPNETCoreDataProtectionAPI** console program and for your own Application.  The encrypted passphrase can only be decrypted on the same server where it was originally encrypted.  Copy the **ProtectSecretsWithASPNETCoreDataProtectionAPI** Console Program (i.e. the entire application folder) run it on each server that needs to perform PGP operations.    Figure : Open File Explorer and browse to the executable file you wish to run as different user. Simply hold down the Shift key and right-click on the executable file, select Run as different user from the context menu.  If there is a need to rotate the SSL Cert that is used to protect the Master key of ASP.NET Core Data Protection API on your Server, simply delete away the existing one from the **Current User’s Personal Cert Store** and rerun the **ProtectSecretsWithASPNETCoreDataProtectionAPI** Console Program to encrypt your secretPassPhrase again. The console program will auto regenerate and re-import a replacement one as it performs your encryption operation.  Once you delete away the old SSL Cert, youwill not be able to decrypt the previously encrypted Secret Passphrases anymore. Thereafter, please configure the newly encrypted passphrase for your own Application that is running on that server. |

1. Configure the appSettings variables in app.config file of the **PgpCombinedCrypto** project.
   * Set the value for **SenderDataFilePath**. You can use this value to supply your own data files for testing. Otherwise, you could just use the default value.
   * Set the value for **PassphraseProtectionMode** to ASPNET\_DPAPI
   * Set the value for **entropy.** Enter a unique value for your application. Ensure you specify the same values for the **ProtectSecretsWithASPNETCoreDataProtectionAPI** Console Program, **PgpCombinedCrypto** and your own Application (i.e. the one that implement gpgme-sharp).
   * Set the value for **SSLCertDistinguishedSubjectName.** Enter a unique value for your application. This is for generating a self-sign SSL Cert that will be automatically created and installed on your development machine/server when you first run this console program on your machine to perform an encryption of a secret passphrase. This SSL Cert will be used to encrypt and decrypt the ASP.NET Core Data Protection Masterkey in your keyring. Ensure you specify the same values for the **ProtectSecretsWithASPNETCoreDataProtectionAPI** Console Program, **PgpCombinedCrypto** and your own Application (i.e. the one that implement gpgme-sharp).
   * Set the value for **AliceEmailAddress.** Enter the email of the sender’s OpenPGP keypair.
   * Set the value for **AliceEncryptedSecretPassPhrase\_ASP\_DPAPI.** Enter the encrypted passphrase generated on the same machine in the preceding step.
   * Set the value for **BobEmailAddress.** Enter the email of the recipient’s OpenPGP keypair.
   * Set the value for **BobEncryptedSecretPassPhrase\_ASP\_DPAPI.** Enter the encrypted passphrase generated on the same machine in the preceding step.

Note: The other appSettings parameters that aren’t mentioned above are irrelevant for this option.

**Tip:** You can still change the values in the app.config after building the application by accessing the **PgpCombinedCrypto.dll.config** file in the application folder.

Setup is complete!

Text

Description automatically generated

Figure : Sample Screenshot of a successful execution of PgpCombinedCrypto.exe