PGPCmd

Table of contents

Introduction

Preface

PGPCmd is a command line utility for performing OpenPGP cryptography. Its syntax is close to the one available in Symantec(r) PGP command line and GnuPG/ gpg

About

The focus of this guide is on implementing PGPCmd as a tool within your overall security system. PGPCmd is only one piece of an overall security system, but it is an extremely important one.

PGPCmd provides encryption, which protects data from the eyes of anyone for whom it was not intended, even those who can see the encrypted data. This protects information from both internal and external “outsiders.”

This Guide describes how to use PGPCmd. It assumes you are familiar with the concepts of public key cryptography

Audience

This guide is designed for system and network administrators who are responsible for their company's security program.

Conventions

The following describes the conventions used in this guide:

Bold, courier font	Pathnames, filenames, syntax, and special keys on the keyboard are shown in a bold, sans-serif font.
angle brackets < > Angle brackets (<>)	indicate a variable. You supply a value of the type indicated.
square brackets [ ]	Square brackets ([]) indicate an option. The value indicated is not required.

Getting help

In order to display help usage screen type:

pgpcmd --help

or

pgpcmd -h

Contacting DidiSoft

Technical support support@didisoft.com

Web: didisoft.com

Phone: +1-256-907-7816

7:00 AM to 8:00 PM (GMT)
7:00 AM to 4:00 PM (EST)

Recommended readings

This section identifies Web sites, books, and periodicals about the history, technical aspects, and politics of cryptography, as well as trusted PGP download sites.

Books

Serious Cryptography: A Practical Introduction to Modern Encryption: by Jean-Philippe Aumasson 2017, ISBN 978-1593278267.

Cryptography (Security and Digital Forensics) by Bill Buchanan (Author) 2017 ISBN: 978-8793379107

Web sites

https://tools.ietf.org/html/rfc4880 - OpenPGP standard

https://tools.ietf.org/html/rfc6637 - Elliptic Curve cryptography in OpenPGP

https://tools.ietf.org/html/draft-ietf-openpgp-rfc4880bis - The latest draft of the OpenPGP standard

• **www.iacr.org -** International Association for Cryptologic Research (IACR). The IACR holds cryptographic conferences and publishes journals.

• **www.nist.gov -** The National Institute of Standards and Technology

PGPCmd basics

PGPCmd is designed to seamlessly integrate into existing business processes to protect your corporate information while in storage or transit. The product's flexible command line interface allows you to quickly integrate PGPCmd with automated processes and web-based applications.

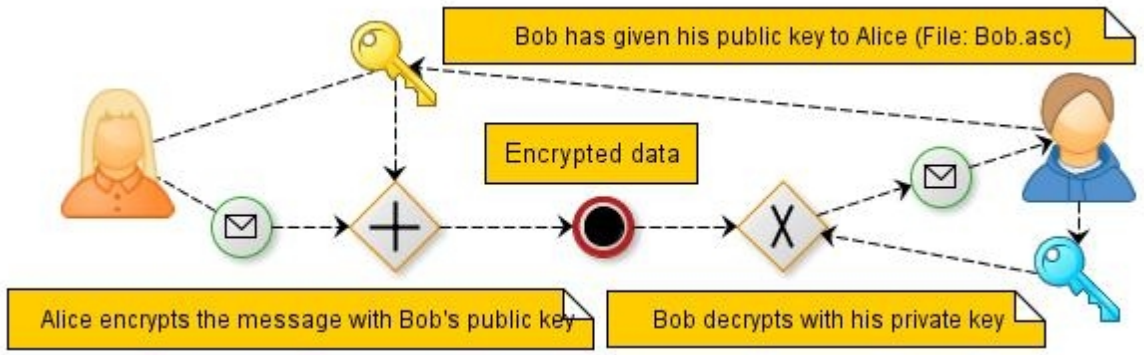
The following are examples how you can use this product to protect your business processes:

A company's Human Resources (HR) group uses E-Business Server to securely send employee records over the Internet to a benefits provider. Prior to sending the records, an automated process on one of the
company's HR servers uses PGPCmd to encrypt the records to the public key corresponding to the company's benefits provider. After the data has been encrypted, the server automatically establishes a connection to the benefits provider and transfers the data. A separate process on the benefits provider's server detects the new files, decrypts them with PGPCmd, and sends them to their final destination.

A circuit board manufacturer shares large, confidential engineering designs with a business partner who is going to manufacture several key components for the board. The manufacturer's server automatically
transfers the designs on a nightly basis via SFTP to the partner's server. By leveraging the benefits of PGPCmd, these companies can now safely transfer these files over the Internet. PGPCmd also compresses data before it encrypts it.

Key concept

PGPCmd is based on a widely accepted and highly trusted public key encryption system, as shown below, in which two complementary keys, called a **key pair**, are used to maintain secure
communications. One of the keys is designated as a **private key** to which only you have access and the other is a **public key** which you freely exchange with other PGPCmd users or users of other OpenPGP implementations. Both your private and your public keys are stored in a **key store**.



Configuration

The program be configured via an XML configuration file with name **pgpcmd.exe.config**

Various settings can be customized including:

1. KeyStore file location
2. pubring.pgp and secring.pgp files reuse
3. Plugins
4. Notification via Email

KeyStore file

OpenPGP keys are stored inside a file called Key store.

The default location of the Key Store is **%PROGRAMDATA%\PGPCmd\key.store**

This can be changed via a configuration key in **%PROGRAMDATA% PGPCmd.exe.config**

Sample configuration

```
<configuration>
  <appSettings>
    <add key="KeyStore" value="C:\Temp\my.keystore" />
  </appSettings>
</configuration>
```

KeyStore as command line parameters

Alternatively the key storage can be provided as a command line parameter

Example:

pgpcmd --key-store C:\Temp\my.keystore

Password protecting

The keyStore can be password protected (*AES-256* bit encryption) by specifying an encryption password

```
<configuration>
  <appSettings>
    <add key="KeyStore" value="C:\Temp\my.keystore" />
    <add key="KeyStorePassword" value="123456" />
  </appSettings>
</configuration>
```

or via the command line

pgpcmd --key-store C:\Temp\my.keystore --key-store-password 123456

Pubring and Secring

PGPCmd can reuse in read-only fashion keys from Symantec's PGP or GnuPG.

In order to reuse such keys we have to specify the location of those files in the application config file **pgpcmd.exe.config**

Sample configuration

```
<configuration>
  <appSettings>
    <add key="PublicKeyRing" value="C:\Users\sales_000\AppData\Roaming\gnupg
\pubring.gpg" />
    <add key="PrivateKeyRing" value="C:\Users\sales_000\AppData\Roaming\gnupg
\secring.gpg" />
  </appSettings>
</configuration>
```

Keyrings as command line parameters

Alternatively they can be provided as command line parameters

Example:

pgpcmd.exe --public-key-ring C:\Users\sales_000\AppData\Roaming\gnupg\pubring.gpg --private-key-ring C:\Users\sales_000\AppData\Roaming\gnupg\secring.gpg

Cryptography operations

Encrypt (-e, --encrypt)

--encrypt (-e)

Encrypts a document to specified recipients. Input is a list of files. Output is an archive.

The usage format is

pgpcmd --encrypt --recipient <user>[;<user2>;<user3>;...] --output <output_file> <input>

<input> is the name of the file to be encrypted. It is required. You can encrypt multiple files by listing them, separated by a space.

<user> is the user ID, portion of the user ID, or the key ID of the recipient. It is required. The public key of the recipient must be on the keyring. You You can encrypt the file to multiple recipients by listing them, separated by ;

Options

-a or --armor armors the encrypted file.

-i or --integrity-protect sets the modification detection code on

-o or --output lets you specify a different name for the encrypted file. Default one is the input file name with extension .pgp

Example usage

pgpcmd.exe -e -r richard@supersafe.org -o data.pgp data.txt

Decrypt (-d, --decrypt)

--decrypt(-d)

Decrypts encrypted data.

Usage:

pgpcmd --decrypt <input>

Parameters

<input> is the name of the file to be decrypted. It is required.

-p or --passphrase Password that unlocks the private decryption key. If not provided is assumed to be an empty password

-o or --output lets you specify a different name for the decrypted file. Default one is the

input file name without the extension .pgp

Sign (-s, --sign)

--sign (-s)

Signs data.

The output is in an unencrypted OpenPGP format with attached signature.

Usage:

pgpcmd --sign --signer <key ID or user ID> --passphrase <passphrase> [-o <output>] <input>

Parameters

<input> is the name of the file to be signed. It is required.

<signer> User ID or key ID of the signing key

-p or --passphrase Password that unlocks the private signing key. If not provided is assumed to be an empty password

-o or --output lets you specify a different name for the decrypted file. Default one is the input file name without the extension .pgp

--signer Signer key ID or user ID

Verify (--verify)

--verify

Verifies the signature of a signed unencrypted OpenPGP file

The usage format is

pgpcmd --verify <input>

<input> is the name of the file to be encrypted. It is required. You can encrypt multiple files by listing them, separated by a space.

Options

<input> input file

Example usage

pgpcmd.exe --verify data.pgp

Sign and Encypt (-e -s, --encrypt --sign)

--encrypt --sign (-e -s)

Signs a document then encrypts it to specified recipient(s). Input is a list of files. Output is an archive.

The usage format is

pgpcmd --encrypt --sign --recipient <user>[;<user2>;<user3>;...] --signer <signer> --passphrase <password> --output <output_file> <input>

<input> is the name of the file to be encrypted. It is required. You can encrypt multiple files by listing them, separated by a space.

<user> is the user ID, portion of the user ID, or the key ID of the recipient. It is required. The public key of the recipient must be on the keyring. You You can encrypt the file to multiple recipients by listing them, separated by ;

Options

<signer> User ID or key ID of the signing key

-p or --passphrase Password that unlocks the private signing key. If not provided is assumed to be an empty password

-a or --armor armors the encrypted file.

-i or --integrity-protect sets the modification detection code on

-o or --output lets you specify a different name for the encrypted file. Default one is the input file name with extension .pgp

Example usage

pgpcmd.exe -e -r richard@supersafe.org -o data.pgp data.txt

Key operations

Import (--import)

Imports public and private keys into the Key Store

--import, -i

The usage format is

pgpcmd --import <input file>

<input> is the name of the file to be imported. It is required.

Options

<input file> Absolute or relative path to a key file to be imported

Example usage

pgpcmd --import mykey.asc

pgp --import mykey.asc

Export (--export)

Exports a public keys from the Key Store

--export

The usage format is

pgpcmd --export <User ID | Key ID>

Options

<User ID | Key ID> User ID or hexadecimal Key ID of the key to be exported

Example usage

pgpcmd --export 0x927865a1 -a -o key1.asc

pgp --export "John Doe" -a -o john.asc

Generate Key (--gen-key)

OpenPGP keys an be created with the --gen-key parameter.

--gen-key

Creates an OpenPGP key inside the key storage.

The usage format is

pgpcmd --gen-key <user ID> --key-type <type> --encryption-bits <bits> [--expiration-date <date>] [--passphrase <password>] [-o <output file>]

<input> is the name of the file to be encrypted. It is required. You can encrypt multiple files by listing them, separated by a space.

Options

<user ID> User ID of the key owner in the format : "*Name <email>"*
<type> key type, Possible values:
rsa - RSA based key
dh - DH/DSS based key
ecc - Elliptic Curves based key

<bits> - key size in bits. For Elliptic Curves keys, values can be 256 (*for NIST P-256*), 384 (*for NIST P-384*), 521 (*for NIST P-521*)
<date> - if the key will have validity period, this is the expiration end date

<output file> - output file for the key pair

Example usage

pgpcmd.exe --gen-key "John Doe <john@gmail.com>" --key-type ECC --encryption-bits 521 --output my_key.asc

List keys (-l, --list-keys)

Lists the keys contained in the Key Store

--list-keys, -l

The usage format is

pgpcmd --list-keys [<User ID | Key ID>]

pgpcmd -l [<User ID | Key ID>]

Options

<User ID | Key ID> (Optional) User ID or hexadecimal Key ID of the key to be listed

Example usage

pgpcmd --list-keys 0x927865a1

pgp -l

Sign Key (--sign-key)

Signs every user ID on a key

--sign-key

The usage format is

pgpcmd/pgp --sign-key <User ID | Key ID> --signer <signer User ID | Key ID> --sig-type <signature type> --passphrase <password>

Options

<User ID | Key ID> User ID or hexadecimal Key ID of the public key to be signed
<signer User ID | Key ID> User ID or hexadecimal Key ID of the signing key
<signature type> One of **local, exportable, meta-introducer, trusted-introducer**, with the following meaning:

- **local** - the signature is non-exportable. It cannot be exported in any way. Use this signature when you believe the key is valid, but you don't want others to rely on your opinion of the key.
- **exportable** - the signature can be exported with the key. Use this signature when you believe the key is valid and you want others to be able to rely on your opinion of the key.
- **meta-introducer** - this is a non-exportable meta-introducer, which means that this key and any keys signed by this key with a trusted introducer validity assertion are fully trusted introducers to you.
- **trusted-introducer** - you certify that this key is valid and that the owner of the key should be completely trusted to vouch for other keys. This signature type is exportable. If you get a key from someone that has been signed by an individual whom you have designated as trustworthy, the key is considered valid even though you have not done the check yourself.

<password> - pass phrase of the signing key

Example usage

pgpcmd/pgp --sign-key 0x927865a1 --signer "My Key" --sig-type meta-introducer --passphrase 123456

Set Trust (--set-trust)

Changing your trust settings on a key

--set-trust

The usage format is

pgpcmd/pgp --set-trust <User ID | Key ID> --trust <trust>

Options

<User ID | Key ID> User ID or hexadecimal Key ID of the public key to be set
<trust> level of trust, one of:

- **none** - if you do not know if you trust the owner of this key to act as a trusted introducer, or if you do not trust the owner of this key.
- **marginal** - if you usually trust the owner of this key to act as a trusted introducer.
- **complete** - if you always trust the owner of this key to act as a trusted introducer.

Example usage

pgpcmd/pgp --set-trust "My Partner Key" --trust complete

Plugins

A Plug-in architecture for logging audit information is available in PGPCmd.

The base of the Plug-in system is an Interface LogInterface, which is defined in a separate assembly DLL *PGPCmd.LogInterface.dll*:

```
namespace PGPCmd
{
    /// <summary>
    /// Base Interface that all plugins must implement
    /// </summary>
    public interface LogInterface
    {
        /// <summary>
        /// Plugin Name, must not contain whitespace and shall exist as app.config
section
        /// </summary>
        string Name { get; }

        /// <summary>
        /// Plugin explanation
        /// </summary>
        string Explanation { get; }

        /// <summary>
        /// Configures the plugin with settings from app.config
        /// </summary>
        /// <param name="configs">config section from loading program app.config</param>
        /// <param name="error">error description</param>
        /// <returns>true on success, false on error initializing</returns>
        bool Init(System.Collections.Specialized.NameValueCollection configs, out String
error);

        /// <summary>
        /// Logs event information
        /// </summary>
        /// <param name="computerName">Computer Name</param>
        /// <param name="userName">current User name</param>
        /// <param name="operation">PGP operation</param>
        /// <param name="fileIn">input file</param>
        /// <param name="fileOut">output file</param>
        /// <param name="principalId">Key Principal (email, id, etc) used in operaton </
param>
        /// <param name="resultCode">0=success, else=failure</param>
        /// <param name="resultDescription">description associated with result</param>
        /// <param name="error">error description</param>
        /// <returns>true on success, false otherwise</returns>
        bool Log(string computerName,
            string userName,
            string operation,
            string fileIn,
            string fileOut,
            string principalId,
            int resultCode,
            string resultDescription,
            out string error);
    }
}
```

Configuration

Plugins must be located in the same folder where PGPCmd is.

Each plugin must also have its own configuration section in the app.config with name - the Name of the Plugin.

For example a Plugin

```
    public class MsSqlLogger : PGPCmd.LogInterface
    {
        private string connectionString;

        /// <summary>
        /// Plugin Name, must not contain whitespace and shall exist as app.config
section
        /// </summary>
        public string Name { get { return "MsSqlLogger";  } }
...
```

must have its configuration section in app.config:

```
  <configSections>
    <section name="MsSqlLogger" type="System.Configuration.NameValueSectionHandler" />
  </configSections>
  <MsSqlLogger>
    <add key="connectionString" value="Data Source=KETLABS3;Initial
Catalog=dotnettest;Integrated Security=True" />
  </MsSqlLogger>
```

Sample Plugin

This is a sample plugin that just prints on the console:

```
    public class ConsoleLogger : PGPCmd.LogInterface
    {
        private string connectionString;

        /// <summary>
        /// Plugin Name, must not contain whitespace and shall exist as app.config
section
        /// </summary>
        public string Name { get { return "ConsoleLogger";  } }

        /// <summary>
        /// Plugin explanation
        /// </summary>
        public string Explanation { get { return "Logs on the console"; } }

        /// <summary>
        /// Configures the plugin with settings from app.config
        /// </summary>
        /// <param name="configs">config section from loading program app.config</param>
        /// <param name="error">error description</param>
        /// <returns>true on success, false on error initializing</returns>
        public bool Init(System.Collections.Specialized.NameValueCollection configs, out
String error)
        {
            return true;
```

```
        }

        /// <summary>
        /// Log event
        /// </summary>
        /// <param name="computerName">Computer Name</param>
        /// <param name="userName">current User name</param>
        /// <param name="operation">PGP operation</param>
        /// <param name="fileIn">input file</param>
        /// <param name="fileOut">output file</param>
        /// <param name="principalId">Key Principal (email, id, etc) used in operaton </
param>
        /// <param name="resultCode">0=success, else=failure</param>
        /// <param name="resultDescription">description associated with result</param>
        /// <param name="error">error description</param>
        /// <returns>true on success, false otherwise</returns>
        public bool Log(string computerName,
            string userName,
            string operation,
            string fileIn,
            string fileOut,
            string principalId,
            int resultCode,
            string resultDescription,
            out string error)
        {
            error = string.Empty;

                StringBuilder stringBuilder = new StringBuilder();
                stringBuilder.Append("@SysID=").Append(computerName).Append
(Environment.NewLine);
                stringBuilder.Append("@UserID=").Append(userName).Append
(Environment.NewLine);
                stringBuilder.Append("@Operation=").Append(operation).Append
(Environment.NewLine);
                stringBuilder.Append("@FileIn=").Append(fileIn).Append
(Environment.NewLine);
                stringBuilder.Append("@FileOut=").Append(fileOut).Append
(Environment.NewLine);
                stringBuilder.Append("@Principal=").Append(principalId).Append
(Environment.NewLine);
                stringBuilder.Append("@KeyID=").Append(principalId).Append
(Environment.NewLine);
                stringBuilder.Append("@ResultCode=").Append(resultCode).Append
(Environment.NewLine);
                stringBuilder.Append("@ResultDesc=").Append(resultDescription).Append
(Environment.NewLine);
                Console.WriteLine(stringBuilder.ToString());

                return true;
        }

    }
```

Configuration

The plugin must be registered in the configuration section like:

```
  <configSections>
    <section name="ConsoleLogger" type="System.Configuration.NameValueSectionHandler" />
  </configSections>
  <ConsoleLogger>
```

```
</ConsoleLogger>
```

MsSqlLogger

MsSqlLogger is a plugin that ships with PGPCmd and is located in **MsSqlLoggerPlugin.dll**.

It logs audit records for each operation in a MS SQL Server database.

The plugin is configured via app.config with:

```
<configSections>
  <section name="MsSqlLogger" type="System.Configuration.NameValueSectionHandler" />
</configSections>

<MsSqlLogger>
  <add key="connectionString" value="...MS SQL connection string here ...." />
</MsSqlLogger>
```

Trust

Trust is confidence in another person's ability to validate a key. If you designate someone a trusted introducer, then all keys validated by the trusted introducer are considered to be valid to you.

If you get a key from someone that has been signed by an individual whom you have designated as trustworthy, the key is considered valid even though you have not done the check yourself.

Error Notifications via Email

A built-in mechanism for Email notifications upon error conditions is also available.

In order to enable the Email notifications on error we have to configure it via PGPCmd.exe.config:

```
<configSections>
  <section name="ErrorNotifier" type="System.Configuration.NameValueSectionHandler" />
</configSections>
<ErrorNotifier>
  <add key="smtpHost" value="" />
  <add key="smtpPort" value="" />
  <add key="smtpUser" value="" />
  <add key="smtpPass" value="" />
  <add key="fromEmail" value="" />
  <add key="toEmail" value="" />
  <add key="subject" value="" />
</ErrorNotifier>
```

Copyright