PdfCrypter COM interface documentation

Introduction

With PdfCrypter you can encrypt your documents to protect them against unauthorized read or modification. There are two ways to encrypt a PDF document.
With ‚Standard Security' you can apply password protection to a PDF document. You can assign a user password to protect against unauthorized read and an owner password and permissions to protect against modification.
With ‚Public Security' you can apply certificate protection to a PDF document. For each recipient of the document the recipient's certificate is added to the document and each recipient can have different access rights for the document.
Another feature is to decrypt PDF documents which are encrypted using standard security or public security. The decrypting process needs the user to authenticate (See Authentication).

Licenses

PdfCrypter comes with three license types.
Demo version which provides full function of the professional version, with the restriction of an annotation added to each page. The annotation shows a hint the document is processed with PdfCrypter.
Standard version provides standard security protection.
Professional version provides standard security and public security protection.

Authentication

Decrypting a PDF document needs the user to authenticate. For documents protected with standard security a password is needed and an OnPassword event is fired. An application using PdfCrypter can define an event handler to prompt for password or return a password from password list. The Event is fired as long as the password is not correct or cancelled (See TPdfCrypter.OnPassword for details). If no OnPassword handler is defined a default dialog appears prompting for a password. For documents protected with public security access to the recipients certificate is necessary. The certificate must be on the local system (where PdfCrypter runs) and must contain the private key. As PdfCrypter accesses the private key, a windows dialog may appear asking the user to allow access or prompting for a password, if the user has a password protected private key.

Class TPdfCrypter

Methods

```
constructor Create(AOwner: TComponent)
```

Description

Create a PdfCrypter object

Input

AOwner: Owner component

```
destructor Destroy;
```

Description
Free PdfCrypter object

```
procedure Connect;
```

Description
Connect to PdfCrypter COM Server.

Condition
PdfCrypter COM Server must be registered on local machine using regsvr32.exe.

```
procedure OpenDocument(const Filename: WideString; out Success: WordBool)
```

Description
open a PDF document for further processing with PdfCrypter.

Input
Filename: name of PDF file.

Output
Success : true, if PDF file could be open.

```
procedure CloseDocument
```

Description
Close PDF document.

Condition
Document must be open.

```
procedure IsDocumentOpen(out IsOpen: WordBool)
```

Description
Test, if a PDF document is opened with PdfCrypter.

Output
IsOpen: true, if document is open.

```
procedure SaveDocumentAs(const Filename: WideString)
```

Description
Save PDF document to file.

Input
Filename: name of target PDF file.

Condition
Document must be open. Filename must be different from filename used in OpenDocument.

```
procedure SaveDocument
```

Description
Save PDF document

```
procedure GetDocumentVersion(out MajorVersion: SYSINT;
                             out MinorVersion: SYSINT)
```

Description
Get PDF document version.

Output
MajorVersion: major PDF version number
MinorVersion: minor PDF version number

Condition
Document must be open.

```
procedure GetDocumentInfo(out Title: PChar; out Producer: PChar;
                          out Author: PChar; out Success: WordBool)
```

Description
Get PDF document information.

Output
Title: Document title
Producer: Application created the PDF document
Author: Author of document
Success: for Unencrypted Documents always true, else true, if authentication was successful.

Condition
Document must be open. SecurityInfo must not be crpUnknown.

Remark
GetDocumentInfo reads part of the PDF Info dictionary, which may be encrypted. In this case authentication is needed (See Authentication). Data of Title, Producer and Author is only valid, when Success is true.

```
procedure GetStandardSecurityInfo(out EncryptMetadata: WordBool;
                                  out CryptAlgorithm: CrpCryptAlgorithm;
```

```
                                out KeyLength: SYSINT;
                                out Permissions: LongWord)
```

Description

Get security information for PDF document encryted with standard security.

Output

EncryptMetadata: true, if PDF metadata is encrypted
CryptAlgorithm: algorithm used for encryption. If PDF document uses crypt filters, the algorithm used for the default stream filter is returned.
KeyLength: Length of encryption key in bits. If PDF document uses crypt filters, the key length for default stream filter is returned.
Permissions: User Access rights for the document (See Permissions for description)

Condition

Document must be open. The value of SecurityInfo property must be crpStandard else information is invalid.

```
procedure GetPublicSecurityInfo(out CryptAlgorithm: CrpCryptAlgorithm;
                                out KeyLength: SYSINT;
                                out EncryptMetadata: WordBool;
                                out ARecipients: IRecipients)
```

Description

Get security information for PDF document encrypted with public security.

Output

CryptAlgorithm: Algorithm used for encryption. If PDF document uses crypt filters, the algorithm used for default stream filter is returned.
KeyLength: Length of encryption key in bits. If PDF document uses cryptfilters, the key length for default stream filter is returned.
EncryptMetadata: true, if PDF metadata is encrypted
ARecipients: collection of recipient data returned as IRecipients interface.

Condition

Document must be open. The value of SecurityInfo property must be crpPublicKey else information is invalid.

```
procedure GetBestEncryptionMethod(
                out EncryptionMethod: CrpEncryptionMethod)
```

Description

Call this to get best (highest) encryption method can be applied to current PDF document, depending on PDF document version.

Output

EncryptionMethod: highest Encryption method can be used with PDF document.

Condition

Document must be open.

```
procedure AddStandardSecurity(const UserPassword: WideString;
                              const OwnerPassword: WideString;
                              Permissions: LongWord;
                              EncryptMetadata: WordBool;
                              EncryptionMethod: CrpEncryptionMethod)
```

Description

Apply Standard Security (password protection) to PDF document. See Remark 2.

Input

UserPassword: Password needed to open PDF document for reading.
OwnerPassword: Password needed to change permissions.
Permissions: Permissions for those gained document access with UserPassword. See Remark 1.
EncryptMetadata: true to Encrypt documents metadata or false to leave unencrypted. See Remark 3.
EncryptionMethod: Encryption method

Condition

Document must be open. EncryptionMethod must not be crmDocumentCompatibly with a PDF 1.0 Document.

```
procedure AddPublicSecurity(const ARecipients: IRecipients;
                            EncryptionMethod: CrpEncryptionMethod;
                            EncryptMetadata: WordBool)
```

Description

Apply Public Security (certificate protection) to PDF document. See Remark 2.

Condition

Document must be open. Apply only to PDF documents with version of at least 1.4.

```
procedure RemoveSecurity
```

Description

Remove Security from PDF document. See Remark 2.

Condition

Document must be open. SecurityType must not be not crpUnknown.

```
procedure GenerateCertificate(CertificateData: OleVariant;
                              out X509Certificate: OleVariant)
```

Description

Generates a self signed X.509 Certificate

Input

CertificateData A ASN.1 structure containing parameters to generate a X.509 certificate. See below.

Output

X509Certificate ASN.1 structure of a X.509 certificate (can be stored as .cer file)

ASN.1 structure for CertificateData

```
CertificateData  ::= Sequence {
  Version       INTEGER,
  Algorithm     INTEGER,
  KeyLength     INTEGER,
  ValidFrom     Datetime (UCT),
  ValidTo       Datetime (UCT),
  SerialNumber  INTEGER,
  Issuer        RDNSequence (see X.501),
  Subject       RDNSequence
}
```

Version: Version number of this structure must be 1.
Algorithm: Signature Algorithm identifier must be a Value of CrpCertAlgorithm.
KeyLength: The length of Encryption key in Dwords.
ValidFrom, ValidTo: Validity period of the certificate.
SerialNumber: serial number of the certificate.
Issuer: Exhibitor properties.
Subject: Applicant properties.

```
procedure SetLicenseInfo(const Username: WideString;
                         const LicenseKey: WideString)
```

Description

Set License information.

Input

Username: user name
LicenseKey: Serial number

```
function Version: BStr;
```

Description

Get the version of the COM server library.

```
function GetHashValue: BStr;
```

Description

Get the hash value for the library file. The hash is built with SHA256 algorithm.
The hash value result is in hexadecimal form.

Remark 1

Permissions are a combination (+) of values of CrpSecurityPermissions. Some Permissions need at least PDF Version 1.4 see CrpSecurityPermissions. If you use these permissions with a PDF document of lower version the version number is set to 1.4. If EncryptionMethod is crmDocumentCompatibly and PDF

version is lower 1.4 then permissions marked with PDF 1.4 are ignored.

Remark 2

Adding security to PDF document or removing security from PDF document does not affect the document until you call SaveDocument or SaveDocumentAs. This may lead to inconsistencies. For example you remove security from a encrypted document and call GetDocumentInfo, which accesses encrypted information, you may asked for a password to decrypt information. An application using PdfCrypter must handle these inconsistencies itself. To Avoid these problems call SaveDocument(As) each time you are changing security and reopen the new document with OpenDocument.

Remark3

Setting EncryptMetadata to false to leave metadata unencrypted is only possible with PDF Version 1.5 or greater. If EncryptionMethod is crmDocumentCompatibly and PDF documents version number is lower then 1.5 then a value of false for EncryptMetadata is ignored. For other values of EncryptionMethod the PDF version number is set to 1.5 if the current document contains a lower version number.

Properties

```
property SecurityInfo: CrpSecurityType readonly
```

Description

Kind of Security used to encrypt PDF document (See CrpSecurityType).

Condition

Document must be open.

```
property Version: WideString readonly
```

Description

Version number of PdfCrypter COM Interface.

```
property LicenseType: CrpLicenseType readonly
```

Description

LicenseType of PdfCrypter (See CrpLicenseType)

```
property Recipients: IRecipients readonly
```

Description

Creates a new Recipient Collection accessible through IRecipients interface.

```
property DocumentPageCount: Integer readonly
```

Description

Number of pages in PDF Document.

Condition
Document must be open.

```
property DaysLeft: Integer readonly
```

Description
The number of days left for using PdfCrypter trial version.

Condition
Available after registration. Information loaded with initialization.

Events

```
property OnProgressTotal : procedure(Sender : TObject; Percent: SYSINT;
                                     out Abort: WordBool)
```

Description
Fired by some methods to show progress of the method call.

Input
Sender: PdfCrypter object (TPdfCrypter)
Percent: Progress in percent

Output
Abort: Set to true, to cancel operation else this can be leaved unchanged.

Remark
Fired by SaveDocument, SaveDocumentAs, OpenDocument.

```
property OnProgressDetail : procedure(Sender : TObject; Percent: SYSINT;
                                      out Abort: WordBool)
```

Description
Fired by some methods to show progress of parts of a method called.

Input
Sender: PdfCrypter Object (TPdfCrypter)
Percent: Progress in percent.

Output
Abort: Set to true, to cancel operation else this can be left unchanged.

Remark
Fired by SaveDocument, SaveDocumentAs, OpenDocument.

```
property OnStatusTotal : procedure(Sender : TObject; Text: OleVariant)
```

Description

Message of action performed currently.

Input

Sender: PdfCrypter object (TPdfCrypter)
Text: Status message (string)

Remark

Fired by SaveDocument, SaveDocumentAs, OpenDocument.

```
property OnStatusDetail : procedure(Sender : TObject; Text: OleVariant)
```

Description

Message containing detailed information while processing a method call.

Input

Sender: PdfCrypter object (TPdfCrypter)
Text: Status message (string)

Remark

Fired by SaveDocument, SaveDocumentAs, OpenDocument.

```
property OnError : procedure(Sender : TObject; Text: OleVariant;
                             var Handled : WordBool)
```

Description

This event is fired, when an error occurs.

Input

Sender: PdfCrypter object (TPdfCrypter)
Text: Error message (string)

Output

Handled: Set to true in your own handler, else an Exception is raised.

```
property OnPassword : procedure(Sender : TObject; out Password: OleVariant;
                                out Cancel: WordBool;
                                var OwnHandler: WordBool)
```

Description

This event is fired, when a PDF document protected with standard security needs to be decrypted.

Input

Sender: PdfCrypter object (TPdfCrypter)

Output

Password: Password (string)
Cancel: Set to true, to cancel operation.
OwnHandler: Set to true, if OnPassword does handle this event, else leave unchanged.

Remark

If OnPassword is not assigned the value of OwnHandler is false. In this case a default dialog appears prompting for password. Fired By SaveDocument, SaveDocumentAs, RemoveSecurity.

IRecipients

Collection of IRecipient items.

Methods

```
procedure Remove(Index: SYSINT)
```

Description

Remove i-th Recipient item from collection.

Input

Index: Index of item to remove.

Condition

Index must be in range of 1 to value of Count property.

```
procedure Clear
```

Description

Clear Collection.

Properties

```
property Count: Integer readonly
```

Description

Number of IRecipient items in collection.

```
property Item[Index: SYSINT]: IRecipient readonly
```

Description

Get i-th Item of collection.

Input

Index: Index of item.

Condition

Index must be in range of 1 to value of Count property.

```
property Add: IRecipient readonly
```

IRecipient

This interface is used to access the collection items of the IRecipients interface. This interface is used for two different purposes. If creating a PDF document encrypted with public security you add Recipients to the IRecipients collection. In this case only the Certificate and the Permissions properties need to be set. If you get information about a PDF document encrypted with public security, only the SubjectData property of IRecipient is filled.

Properties

```
property Certificate: OleVariant read/write
```

Description

Certificate is a string containing the DER encoded ASN.1 structure of a X.509 Certificate.

```
property Permissions: LongWord read/write
```

Description

Permissions are a combination (+) of CrpSecurityPermissions values.

```
property SubjectData: OleVariant read/write
```

Description

SubjectData is a string containing information on the applicant of the certificate stored as ASN.1 structure of type IssuerAndSerialNumber specified in RFC 2315.

CrpSecurityType

This type lists the possible security types used with PDF documents:

crpNone: Document is not encrypted
crpStandard : Document is protected with standard security
crpPublicKey: Document is protected with public security
crpUnknown: Document is protected with some security no supported by PdfCrypter

CrpCryptAlgorithm

This type lists encryption algorithms used in PDF documents
craCR4: RC4 Algorithm (propietary algorithm of RSA Security)
craAES: AES Algorithm (Advanced Encryption Standard)

CrpEncryptionMethod

This type lists the encryption methods can be used to encrypt a PDF document with PdfCrypter and the PDF version that is at least needed.
crmNone: No Encryption possible (PDF 1.0)
crmRc4_40: RC4 Algorithm with 40-bit key. (PDF 1.1)
crmRc4_128: RC4 Algorithm with 128-bit key. (PDF 1.4)

crmAES_128: AES Algorithm with 128-bit key. (PDF 1.6)
crmDocumentCompatibly: One of the above depending on version of current PDF document loaded with OpenDocument.

CrpCertAlgorithm

This type list the signature algorithms used in certificate generation
ca_SHA1_RSA: RSA Encryption with SHA-1 finger print.

CrpSecurityPermissions

This type lists the access rights can be applied to a PDF document
crsPrint: print document.
crsModifyContent:
crsExtractText
crsModifiyAnnotations

The following are available with PDF version 1.4 or higher
crsFillFormFields
crsExtractTextAndGraphic
crsAssembleDocument
crsPrintHighQuality

CrpLicenseType

This type lists the possible license modes of PdfCrypter
crlUnregistered: Unregistered (Demo version).
crlStandard: Standard version.
crlProfessional: Professional version.

Example

Adding public security to a PDF document.

```
procedure Example;
var
  Recipients: IRecipients;
  Recipient : IRecipient;
  PdfCrypter : TPdfCrypter;
  IsOpen : Wordbool;
  Certificate : string;
  Permissions : LongWord;
begin
  // create crypter object
  PdfCrypter := TPdfCrypter.Create(nil);
  // connect to com server
  PdfCrypter.Connect;
  // open PDF document to encrypt
  PdfCrypter.OpenDocument('C:\Test.pdf', IsOpen);
  if not IsOpen then
    Exit;
  // get a recipient collection;
  Recipients := PdfCrypter.Recipients;
  // add one recipient to collection
  Recipient := Recipients.Add;
  // load X.509 certificate from a file or from a certificate storage
```

```
  Certificate := ...
  // Set permissions for recipient
  Permissions := crsPrint + crsExtractText;
  Recipient.Certificate := Certificate;
  Recipient.Permissions := Permissions;
  // add certificate protection, encrypt with AES 128 bit
  // do not encrypt metadata
  PdfCrypter.AddPublicSecurity(Recipients, crmAES_128, false);
  // save encrypted document to a new! file.
  PdfCrypter.SaveDocumentAs('C:\Test2.pdf');
  // free crypter object
  PdfCrypter.Free;
  // recipients collection is freed automatically
  // when leaving the procedure.
end;
```