
**Identification cards — Integrated circuit
cards —**

Part 15:

Cryptographic information application

**AMENDMENT 1: Examples of the use
of the cryptographic information application**

Cartes d'identification — Cartes à circuit intégré —

Partie 15: Application des informations cryptographiques

*AMENDEMENT 1: Exemples d'emploi de l'application des informations
cryptographiques*

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2007

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Amendment 1 to ISO/IEC 7816-15:2004 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 17, *Cards and personal identification*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 7816-15:2004/AMD1:2007

Identification cards — Integrated circuit cards —

Part 15:

Cryptographic information application

AMENDMENT 1: Examples of the use of the cryptographic information application

Insert the following new annex after Annex D.

Annex E (informative)

Examples of the use of the cryptographic information application

E.1 Introduction

The purpose of this informative annex is to provide practical examples of the use of the cryptographic information application. By providing sample program code for each example, programmers can see the programmatic connection between high-level ASN.1 representations and low-level BER representations and thus create more efficient and more compact software that uses the cryptographic information application.

Each clause in the annex is a free-standing example and consists of four paragraphs:

1. Description of the example
2. A specification of the example described in paragraph (1) in commented ISO/IEC 7816-15 ASN.1 constructs, using the formal value notation defined in ISO/IEC 8824-1.
3. Annotated code in the ISO/IEC 9899 TC2 C programming language for BER encoding and decoding according to the ASN.1 specification of paragraph (2).
4. BER encoding of the example as produced by the encoder of paragraph (3). Two examples also include graphic representations of the BER at the end of the Annex.
5. The source code provided in paragraph (3) was compiled and run to generate the output shown in paragraph (4).

A transcription of the ASN.1 encoding of the Cryptographic Information Application listed in Annex A above was used for all examples. A free, publically-available ASN.1 compiler was used to generate the BER encoders and decoders from this ASN.1.

E.2 Encoding of a Private Key

E.2.1 Cryptographic Information Application Example Description

This is an example of an ISO/IEC 7816-15 RSA private key.

E.2.2 ASN.1 Encoding of an RSA Private Key

```
privateKeys objects { -- SEQUENCE OF --
  privateRSAKey { -- SEQUENCE --
    commonObjectAttributes { -- SEQUENCE --
      label '4b455931'H -- "KEY1" --,
      flags '80'H,
      authId '41444d'H -- "ADM" --,
      userConsent 1
    },
    classAttributes { -- SEQUENCE --
      id '9b'H,
      usage '2040'H,
      native TRUE,
      accessFlags '98'H,
      keyReference 10
    },
    subClassAttributes { -- SEQUENCE --
      keyIdentifiers { -- SEQUENCE OF --
        { -- SEQUENCE --
          idType 5,
          idValue '3132333435363738'H -- "12345678" --
        }
      }
    },
    typeAttributes { -- SEQUENCE --
      value indirect path { -- SEQUENCE --
        efidOrPath '3f004041'H
      },
      modulusLength 1024
    }
  }
}
```

E.2.3 Code Encoding and Decoding BER from the ASN.1

```
/*
** Encoding of a Private Key as a Data Object in EF.OD
*/
void Part15PrivateKey(const char *label,
  unsigned char objectFlags,
  unsigned char *authId,
  unsigned int authIdLength,
  unsigned int userConsent,
  unsigned char native,
  unsigned char *id, unsigned int idLength,
  unsigned short usageFlags,
  unsigned char accessFlags,
  unsigned int keyReference,
  unsigned int identifierType,
  unsigned char *externalIdentifier,
  unsigned char *path, unsigned int pathLength,
  unsigned int modulusLength
)
{
  unsigned int l;
  CIOChoice *cio;
  PrivateKeyChoice *prk, **prkp;
  CredentialIdentifier *crid, **cridp;

  PrivateKeyObject_PrivateRSAKeyAttributes pattr = { 0 };
  CommonObjectAttributes commonObjAttr = { 0 };
  CommonKeyAttributes commonKeyAttr = { 0 };
  CommonPrivateKeyAttributes commonPrivateKeyAttr = { 0 };
  PrivateRSAKeyAttributes privateRSAKeyAttr = { 0 };
```

```

Path pathOctets                = { 0 };
AsnOcts issuerHash             = { 0 };

char commonObjectFlags[1] = { 0 };
AsnBits commonFlagsAsnBits = { 3, commonObjectFlags };

char keyUsage[2] = { 0 };
AsnBits keyUsageAsnBits = { 10, keyUsage };

char keyAccessFlags[1] = { 0 };
AsnBits keyAccessFlagsAsnBits = { 5, keyAccessFlags };

/*
** Section 8.3 The CIOChoice type
**
** "EF.OD shall contain the concatenation of 0, 1, or more DER-encoded CIOChoice values."
**
*/
cio = (CIOChoice *)calloc(1, sizeof(PublicKeyChoice));

cio->choiceId = CIOCHOICE_PRIVATEKEYS;

/*
** "It is expected that an EF.OD entry will usually reference a separate file (the path
** choice of PathOrObjects) containing CIOs of the indicated type. An entry may, however,
** hold CIOs directly (the objects choice of PathOrObjects), if the objects and the EF.OD
** file have the same access control requirements."
**
** PathOrObjects{PrivateKeyChoice}
**
*/
cio->a.privateKeys = (PrivateKeyChoice *)calloc(1, sizeof(PublicKeyChoice));
cio->a.privateKeys->choiceId = PATHOROBJECTS_PRIVATEKEYCHOICE_OBJECTS;
cio->a.privateKeys->a.objects = AsnListNew(sizeof(void*));

/*
** Section 8.4.1 PrivateKeyChoice
**
** "This type contains information pertaining to a private key. Each value
** consists of attributes common to any object, any key, any private key,
** and attributes particular to the key."
**
*/
prkp = (PrivateKeyChoice **)AsnListAppend(cio->a.privateKeys->a.objects);

*prkp = prk = calloc(1, sizeof(PublicKeyChoice));

prk->choiceId = PRIVATEKEYCHOICE_PRIVATERSAKEY;

prk->a.privateRSAKey = &patrr;

patrr.commonObjectAttributes = &commonObjAttr;
patrr.classAttributes        = &commonKeyAttr;
patrr.subClassAttributes     = &commonPrivateKeyAttr;
patrr.typeAttributes         = &privateRSAKeyAttr;

/*
** Section 8.2.8 CommonObjectAttributes
**
** "This type is a container for attributes common to all CIOs."
**
*/
commonObjAttr.label.octs = _strdup(label);
commonObjAttr.label.octetLen = strlen(label);

commonObjectFlags[0] = objectFlags;
commonObjAttr.flags = commonFlagsAsnBits;

commonObjAttr.authId.octetLen=authIdLength;
commonObjAttr.authId.octs = authId;

commonObjAttr.userConsent = &userConsent;

/*
** Section 8.2.9 CommonKeyAttributes
**
** "The iD field shall be unique for each key information object, except when a public
** key information object and its corresponding private key object are stored on
** the same card. In this case, the information objects shall share the same
** identifier (which may also be shared with one or several certificate information

```

```

** objects ..."
*/
commonKeyAttr.iD.octs = iD;
commonKeyAttr.iD.octetLen = iDLength;

keyUsage[0] = (unsigned char) (usageFlags>>8);
keyUsage[1] = (unsigned char) (usageFlags);
commonKeyAttr.usage = keyUsageAsnBits;

keyAccessFlags[0] = accessFlags;
commonKeyAttr.accessFlags= keyAccessFlagsAsnBits;

commonKeyAttr.native = &native;

commonKeyAttr.keyReference = &keyReference;

/*
** Section 8.2.10 CommonPrivateKeyAttributes
**
** "The name field, when present, names the owner of the key, as specified in a
** corresponding certificate's subject field.
**
** Values of the keyIdentifiers field can be matched to identifiers from external
** messages or protocols to select the appropriate key to a given operation."
**
*/
commonPrivateKeyAttr.keyIdentifiers =
    (CommonPrivateKeyAttributesSeqOf *)AsnListNew(sizeof (void*));

cridp = (CredentialIdentifier **)AsnListAppend(commonPrivateKeyAttr.keyIdentifiers);

*cridp = crid = (CredentialIdentifier *)calloc(1, sizeof(CredentialIdentifier));

issuerHash.octs = _strdup(externalIdentifier);
issuerHash.octetLen = strlen(externalIdentifier);

crid->idType = identifierType;
crid->idValue.value = &issuerHash;
SetAnyTypeByInt(&(crid->idValue), identifierType);

/*
** Section 8.4.2 Private RSA Key Attributes
**
** "PrivateRSAKeyAttributes.value: The value shall be a path to a file containing
** a private RSA key. If there is no need to specify a path to a file, the path
** value may be set to the empty path."
**
*/
privateRSAKeyAttr.value = (ObjectValue *)calloc(1, sizeof(ObjectValue));
privateRSAKeyAttr.value->choiceId = OBJECTVALUE_INDIRECT;
privateRSAKeyAttr.value->a.indirect =
    (ReferencedValue *)calloc(1, sizeof(ReferencedValue));

privateRSAKeyAttr.value->a.indirect->choiceId = REFERENCEDVALUE_PATH;

pathOctets.efidOrPath.octs = (char *)calloc(1, pathLength);
memcpy(pathOctets.efidOrPath.octs, path, pathLength);
pathOctets.efidOrPath.octetLen = pathLength;
privateRSAKeyAttr.value->a.indirect->a.path = &pathOctets;

privateRSAKeyAttr.modulusLength = modulusLength;

/*
** Print the Private Key Data Object
**
*/
PrintCChoice(stdout, cio, 3);

/*
** BER Encode the Private Key Data Object
**
*/
BERLength = BEncCChoiceContent(gb, cio);
}

/*
** Decoding of a Private Key as a Data Object in EF.OD
**
*/
PrivateKeyObject_PrivateRSAKeyAttributes *PrivateRSAKey(unsigned char *BER, unsigned int BERLength)
{

```



```

SBuf b;
GenBuf *gb;
unsigned int bytesDecoded = 0;
ENV_TYPE env;
CIOChoice *cio;
AsnTag tagId0;
AsnLen elmtLen0;

if(setjmp(env) != 0) exit(0);

cio = calloc(1, sizeof(CIOChoice));

SBufInstallData(&b, BER, BERLength);
SBufToGenBuf(&b, &gb);

tagId0 = BDecTag(gb, &bytesDecoded, env);
elmtLen0 = BDecLen(gb, &bytesDecoded, env);

/*
** Decode the RSA Private Key Data Object
*/
BDecCIOChoiceContent(gb, tagId0, elmtLen0, cio, &bytesDecoded, env);

return ((PrivateKeyChoice *) (cio->a.privateKeys->a.objects->first->data))->a.privateRSAKey;
}

```

E.2.4 BER Encoding

```

<EF_OD>
0xa0,0x51,0xa0,0x4f,0x30,0x4d,0x30,0x12,0x06,0x04,0x4b,0x45,0x59,0x31,0x03,0x02,
0x05,0x80,0x04,0x03,0x41,0x44,0x4d,0x02,0x01,0x01,0x30,0x12,0x04,0x01,0x9b,0x03,
0x03,0x06,0x20,0x40,0x01,0x01,0xff,0x03,0x02,0x03,0x98,0x02,0x01,0x0a,0xa0,0x13,
0x30,0x11,0xa0,0x0f,0x30,0x0d,0x02,0x01,0x05,0x04,0x08,0x31,0x32,0x33,0x34,0x35,
0x36,0x37,0x38,0xa1,0x0e,0x30,0x0c,0x30,0x06,0x04,0x04,0x3f,0x00,0x40,0x41,0x02,
0x02,0x04,0x00
</EF_OD>

```

Table E.1 is a diagrammatic representation of this BER encoding.

Private Key data object			
Private Key Choice: Private RSA Key			
4D	Private RSA Key object		
30	12	Common object Attribute	
		0C	04 label 4B, 45, 59, 31
		03	02 flags 05, 80
		04	03 auth Id 41, 44, 44
		02	01 userConsent 01
30	12	Common Key Attribute	
		04	01 iD 9B
		03	03 usage 06, 20, 40
		01	01 native FF
		03	02 accessFlags 03, 98
		02	01 keyReference 0A

Private Key data object			
Private Key Choice: Private RSA Key			
4D	Private RSA Key object		
30	12	Common object Attribute	
		0C 04 label	4B, 45, 59, 31
		03 02 flags	05, 80
		04 03 auth Id	41, 44, 44
		02 01 userConsent	01
30	12	Common Key Attribute	
		04 01 iD	9B
		03 03 usage	06, 20, 40
		01 01 native	FF
		03 02 accessFlags	03, 98
		02 01 keyReference	0A

Table E.1 (continued)

A0	13	Common Private Key Attribute									
		30	11	Sequence							
		A0	0F	keyIdentifier							
				30	0D	Sequence					
						02	01	idType	05		INTEGER
						60	08	idValue	31, 32, 33, 34, 35, 36, 37, 38		OpenType
A1	0e	Private RSA key attribute									
		30	0C	Sequence							
				30	06	Path					
				04	04	efidOrPath			3F, 00, 40, 41		OCTET STRING
		02	02	modulusLength					04, 00		INTEGER

E.3 Encoding of a Protected Data Container

E.3.1 Cryptographic Information Application Example Description

A data container object with two security conditions, one for READ and one for UPDATE. The data in the data container is a BER-TLV. The secret key SK-1 must be verified in order to change password AO-1.

E.3.2 ASN.1 Encoding of the Protected Data Container Object

```
dataContainerObjects objects { -- SEQUENCE OF --
    iso7816DO { -- SEQUENCE --
        commonObjectAttributes { -- SEQUENCE --
            label '444f2d31'H -- "DO-1" --,
            flags '40'H,
            accessControlRules { -- SEQUENCE OF --
                { -- SEQUENCE --
                    accessMode '80'H,
                    securityCondition or { -- SEQUENCE OF --
                        authId '414f2d31'H -- "AO-1" --,
                        authId '414f2d32'H -- "AO-2" --
                    }
                },
                { -- SEQUENCE --
                    accessMode '40'H,
                    securityCondition and { -- SEQUENCE OF --
                        authId '414f2d31'H -- "AO-1" --,
                        authId '414f2d32'H -- "AO-2" --
                    }
                }
            },
            classAttributes { -- SEQUENCE --
            },
            typeAttributes direct '80020102'H
        }
    }
}

authObjects objects { -- SEQUENCE OF --
    pwd { -- SEQUENCE --
        commonObjectAttributes { -- SEQUENCE --
            label '414f2d31'H -- "AO-1" --,
            flags '40'H
        },
        classAttributes { -- SEQUENCE --
            authId '414f2d31'H -- "AO-1" --,
            authReference 1,
            seIdentifier 2
        },
        typeAttributes { -- SEQUENCE --
            pwdFlags '0400'H,
            pwdType 1,
            minLength 4,
            storedLength 12,
            maxLength 8,
            padChar 'ff'H -- " " --,
            path { -- SEQUENCE --
                efidOrPath '3f004045'H
            }
        }
    }
}
```

```

    }
  }
}

secretKeys objects { -- SEQUENCE OF --
  genericSecretKey { -- SEQUENCE --
    commonObjectAttributes { -- SEQUENCE --
      label '534b2d31'H -- "SK-1" --,
      flags '40'H,
      authId '414f2d31'H -- "AO-1" --
    },
    classAttributes { -- SEQUENCE --
      iD '534b2d31'H -- "SK-1" --,
      usage '0200'H,
      native TRUE,
      accessFlags '10'H,
      keyReference 10
    },
    subclassAttributes { -- SEQUENCE --
      keyLen 64
    },
    typeAttributes { -- SEQUENCE --
      keyType {2 8},
      keyAttr '58'H
    }
  }
}

```

E.3.3 Code from the ASN.1 for Encoding and Decoding BER

```

/*
** Encoding of a Protected Data Object
*/
void DataObject(unsigned char *label,
                unsigned char objectFlags,
                unsigned char *password1,
                unsigned char *password2
                )
{
  CIOChoice *cio;
  DataContainerObjectChoice *dco, **dco;
  AccessControlRule *acr, **acrp;
  SecurityCondition *sc, **scp;

  SecurityCondition securityCondition1;
  SecurityCondition securityCondition2;
  AsnOcts authId1;
  AsnOcts authId2;

  CommonObjectAttributes commonObjectAttr = { 0 };
  CommonDataContainerObjectAttributes commonDataContainerObjectAttributes = { 0 };
  ISO7816DOAttributes iso7816DOAttributes = { 0 };

  DataContainerObject_ISO7816DOAttributes patrr = { 0 };

  CredentialIdentifier credentialIdentifier = { 0 };
  Path pathOctets = { 0 };
  AsnOcts dataObjectValue1 = { sizeof(doValue1), doValue1 };

```

```

char commonObjectFlags[1]                = { 0 };
AsnBits commonFlagsAsnBits                = { 2, commonObjectFlags };

char accessControlRuleFlags1[1]           = { 0 };
AsnBits accessControlRuleAsnBits1         = { 4, accessControlRuleFlags1 };
char accessControlRuleFlags2[1]           = { 0 };
AsnBits accessControlRuleAsnBits2         = { 4, accessControlRuleFlags2 };

char usageFlagBits[2]                     = { 0 };
AsnBits usageFlagsAsnBits                 = { 10, usageFlagBits };

authId1.octetLen = strlen(password1);
authId1.octs = strdup(password1);

authId2.octetLen = strlen(password2);
authId2.octs = strdup(password2);

/*
** Data Object Choice
*/
cio = (CIOChoice *)calloc(1, sizeof(DataContainerObjectChoice));
cio->choiceId = CIOCHOICE_DATACONTAINEROBJECTS;
cio->a.dataContainerObjects = (DataContainerObjects *)calloc(1, sizeof(DataContainerObjects));
cio->a.dataContainerObjects->a.objects = AsnListNew(sizeof (void*));
cio->a.dataContainerObjects->choiceId = PATHROBJECTS_DATACONTAINEROBJECTCHOICE_OBJECTS;

dco = (DataContainerObjectChoice **)AsnListAppend(cio->a.dataContainerObjects->a.objects);

*dco = dco = calloc(1, sizeof(DataContainerObjectChoice));

dco->choiceId = DATACONTAINEROBJECTCHOICE_ISO7816DO;

dco->a.iso7816DO = &pattr;

pattr.commonObjectAttributes = &commonObjectAttr;
pattr.classAttributes        = &commonDataContainerObjectAttributes;
pattr.subClassAttributes     = NULL;
pattr.typeAttributes         = &iso7816DOAttributes;

/*
** Common Object Attributes
*/
commonObjectAttr.label.octetLen = strlen(label);
commonObjectAttr.label.octs = label;

commonObjectFlags[0] = objectFlags;
commonObjectAttr.flags = commonFlagsAsnBits;

commonObjectAttr.accessControlRules = AsnListNew(sizeof (void*));

acrp = (AccessControlRule **)AsnListAppend(commonObjectAttr.accessControlRules);
*acrp = acr = calloc(1, sizeof(AccessControlRule));
accessControlRuleFlags1[0] = (unsigned char)(READ_FLAG);
acr->accessMode = accessControlRuleAsnBits1;
securityCondition1.choiceId = SECURITYCONDITION_SEACOS_OR;
securityCondition1.a.seacos_or = AsnListNew(sizeof (void*));
scp = (SecurityCondition **)AsnListAppend(securityCondition1.a.seacos_or);
*scp = sc = calloc(1, sizeof(SecurityCondition));
sc->choiceId = SECURITYCONDITION_AUTHID;
sc->a.authId = &authId1;

```

```

scp = (SecurityCondition **)AsnListAppend(securityCondition1.a.seacos_or);
*scp = sc = calloc(1, sizeof(SecurityCondition));
sc->choiceId = SECURITYCONDITION_AUTHID;
sc->a.authId = &authId2;
acr->securityCondition = &securityCondition1;

acrp = (AccessControlRule **)AsnListAppend(commonObjectAttr.accessControlRules);
*acrp = acr = calloc(1, sizeof(AccessControlRule));
accessControlRuleFlags2[0] = (unsigned char)(UPDATE_FLAG);
acr->accessMode = accessControlRuleAsnBits2;
securityCondition2.choiceId = SECURITYCONDITION_SEACOS_AND;
securityCondition2.a.seacos_and = AsnListNew(sizeof(void*));
scp = (SecurityCondition **)AsnListAppend(securityCondition2.a.seacos_and);
*scp = sc = calloc(1, sizeof(SecurityCondition));
sc->choiceId = SECURITYCONDITION_AUTHID;
sc->a.authId = &authId1;
scp = (SecurityCondition **)AsnListAppend(securityCondition2.a.seacos_and);
*scp = sc = calloc(1, sizeof(SecurityCondition));
sc->choiceId = SECURITYCONDITION_AUTHID;
sc->a.authId = &authId2;
acr->securityCondition = &securityCondition2;

/*
** Common Data Container Object Attributes
*/

/*
** ISO/IEC 7816 Data Object Attributes
*/
iso7816DOAttributes.choiceId = OBJECTVALUE_DIRECT;

iso7816DOAttributes.a.direct.value = &dataObjectValue1;
SetAnyTypeByInt(&(iso7816DOAttributes.a.direct), issuerKeyHash);

/*
** Print the Data Object
*/
PrintCIOChoice(stdout, cio, 3);

/*
** BER Encode the Data Object
*/
BERLength += BEncCIOChoiceContent(gb, cio);
}

void Password(const char *label,
              unsigned char objectFlags,
              unsigned char *authId,
              unsigned int authReference,
              unsigned int seReference,
              unsigned char *id,
              unsigned char *path, unsigned int pathLength,
              unsigned short pwdFlags,
              unsigned int pwdType,
              unsigned int minimumLength,
              unsigned int storedLength,
              unsigned int maximumLength,
              unsigned char paddingCharacter
              )
{

```

```

CIOChoice *cio;
AuthenticationObjectChoice *auth, **authp;

AuthenticationObject_PasswordAttributes pattr                = { 0 };

CommonObjectAttributes commonObjAttr                        = { 0 };
CommonAuthenticationObjectAttributes commonAuthenticationObjectAttr = { 0 };
PasswordAttributes passwordAttributes                       = { 0 };

Path pathOctets                                             = { 0 };
AsnOcts padChar                                            = { 0 };

char commonObjectFlags[1]                                  = { 0 };
AsnBits commonFlagsAsnBits                                = { 2, commonObjectFlags };
char passwordFlags[2]                                      = { 0 };
AsnBits passwordFlagsBits                                  = { 12, passwordFlags };

/*
** Authentication Object Choice
*/
cio = (CIOChoice *)calloc(1, sizeof(AuthenticationObjectChoice));
cio->choiceId = CIOCHOICE_AUTHOBJECTS;
cio->a.authObjects = (AuthObjects *)calloc(1, sizeof(AuthObjects));
cio->a.authObjects->choiceId = PATHROBJECTS_AUTHENTICATIONOBJECTCHOICE_OBJECTS;
cio->a.authObjects->a.objects = AsnListNew(sizeof(void*));

authp = (AuthenticationObjectChoice **)AsnListAppend(cio->a.authObjects->a.objects);

*authp = auth = calloc(1, sizeof(AuthenticationObjectChoice));

auth->choiceId = AUTHENTICATIONOBJECTCHOICE_PWD;

auth->a.pwd = &pattr;

pattr.commonObjectAttributes = &commonObjAttr;
pattr.classAttributes        = &commonAuthenticationObjectAttr;
pattr.subClassAttributes     = (AsnNull *)NULL;
pattr.typeAttributes         = &passwordAttributes;

/*
** Common Object Attributes
*/
commonObjAttr.label.octs = _strdup(label);
commonObjAttr.label.octetLen = strlen(label);

commonObjectFlags[0] = objectFlags;
commonObjAttr.flags = commonFlagsAsnBits;

/*
** Common Authentication Object Attributes
*/
commonAuthenticationObjectAttr.authId.octs = iD;
commonAuthenticationObjectAttr.authId.octetLen = strlen(iD);

commonAuthenticationObjectAttr.authReference = &authReference;
commonAuthenticationObjectAttr.seIdentifier = &seReference;

/*
** Password Attributes
*/

```



```

passwordFlags[0] = (unsigned char) (pwdFlags>>8);
passwordFlags[1] = (unsigned char) (pwdFlags);
passwordAttributes.pwdFlags = passwordFlagsBits;

passwordAttributes.pwdType      = pwdType;
passwordAttributes.minLength    = minimumLength;
passwordAttributes.storedLength = storedLength;
passwordAttributes.maxLength    = (AsnInt *)calloc(1, sizeof(AsnInt));
*passwordAttributes.maxLength   = maximumLength;

padChar.octetLen = 1;
padChar.octs = (char *)calloc(1,1);
padChar.octs[0] = paddingCharacter;
passwordAttributes.padChar = padChar;

pathOctets.efidOrPath.octs = path;
pathOctets.efidOrPath.octetLen = pathLength;
passwordAttributes.path = &pathOctets;

/*
** Print the Authentication Data Object
*/
fprintf(stdout, "\n\n");
PrintCIOChoice(stdout, cio, 3);

/*
** BER Encode the Authentication Data Object
*/
BERLength += BEncCIOChoiceContent(gb,cio);
}

void SecretKey(const char *label,
               unsigned char objectFlags,
               unsigned char *authId,
               unsigned short usageFlags,
               unsigned int keyReference,
               unsigned char *id,
               unsigned int keyLength
              )
{
    CIOChoice *cio;
    SecretKeyChoice *sk, **skp;

    SecretKeyObject_GenericKeyAttributes pattr = { 0 };

    CommonObjectAttributes commonObjAttr = { 0 };
    CommonKeyAttributes commonKeyAttr = { 0 };
    CommonSecretKeyAttributes commonSecretKeyAttr = { 0 };
    GenericKeyAttributes genericKeyAttr = { 0 };

    Path pathOctets = { 0 };
    AsnOcts keyOidOcts = { 0 };
    AsnOcts keyAttrOcts = { 0 };
    AsnOid keyOid = { 0 };
    char commonObjectFlags[1] = { 0 };
    AsnBits commonFlagsAsnBits = { 2, commonObjectFlags };
    char keyUsage[2] = { 0 };
    AsnBits keyUsageAsnBits = { 9, keyUsage };
    char keyNativeAsnBool = FALSE;
    char keyAccessFlags[1] = { 0 };
    AsnBits keyAccessFlagsAsnBits = { 4, keyAccessFlags };

```

```

/*
** Secret Key Choice
*/
cio = (CIOChoice *)calloc(1, sizeof(SecretKeyChoice));
cio->choiceId = CIOCHOICE_SECRETKEYS;
cio->a.secretKeys = (SecretKeys *)calloc(1, sizeof(SecretKeys));
cio->a.secretKeys->choiceId = PATHOROBJECTS_SECRETKEYCHOICE_OBJECTS;
cio->a.secretKeys->a.objects = AsnListNew(sizeof (void*));

skp = (SecretKeyChoice **)AsnListAppend(cio->a.secretKeys->a.objects);

*skp = sk = calloc(1, sizeof(SecretKeyChoice));

sk->choiceId = SECRETKEYCHOICE_GENERICSECRETKEY;

sk->a.genericSecretKey = &pattr;

pattr.commonObjectAttributes = &commonObjAttr;
pattr.classAttributes         = &commonKeyAttr;
pattr.subClassAttributes      = &commonSecretKeyAttr;
pattr.typeAttributes          = &genericKeyAttr;

/*
** Common Object Attributes
*/
commonObjAttr.label.octs = label;
commonObjAttr.label.octetLen = strlen(label);

commonObjAttr.authId.octetLen = strlen(authId);
commonObjAttr.authId.octs = authId;

commonObjectFlags[0] = objectFlags;
commonObjAttr.flags = commonFlagsAsnBits;

/*
** Common Key Attributes
*/
commonKeyAttr.iD.octs = iD;
commonKeyAttr.iD.octetLen = strlen(iD);

keyUsage[0] = (unsigned char)(usageFlags>>8);
keyUsage[1] = (unsigned char)(usageFlags);
commonKeyAttr.usage = keyUsageAsnBits;

keyNativeAsnBool = TRUE;
commonKeyAttr.native = &keyNativeAsnBool;

keyAccessFlags[0] = NEVEREXTRACTABLE_FLAG;
commonKeyAttr.accessFlags = keyAccessFlagsAsnBits;

commonKeyAttr.keyReference = &keyReference;

/*
** Common Secret Key Attributes
*/
commonSecretKeyAttr.keyLen = (AsnInt *)calloc(1, sizeof(AsnInt));

*commonSecretKeyAttr.keyLen = keyLength;

/*

```

```

** Generic Secret Key Type
*/
keyOidOcts.octetLen = 1;
keyOidOcts.octs = (char *)calloc(1,1);
keyOidOcts.octs[0] = 88;
genericKeyAttr.keyType = keyOidOcts;

SetAnyTypeByInt(&genericKeyAttr.keyAttr, subjectKeyId);
keyAttrOcts.octetLen = 1;
keyAttrOcts.octs = (char *)calloc(1,1);
keyAttrOcts.octs[0] = 88;
genericKeyAttr.keyAttr.value = &keyAttrOcts;

/*
** Print the Secret Key Data Object
*/
fprintf(stdout, "\n\n");
PrintCIOChoice(stdout, cio, 3);

/*
** BER Encode the Secret Key Data Object
*/
BERLength += BEncCIOChoiceContent(gb,cio);
}

/*
** Finding the Authentication Object that Protects a Data Object
*/
void Access_Condition_for_Data_Object(unsigned char *DOBER, unsigned int DOBERLength,
                                     unsigned char *AOBER, unsigned int AOBERLength)
{
    ENV_TYPE env;
    CIOChoice *cioDO, *cioAO;
    SBuf dob, aob;
    GenBuf *dogb, *aogb;
    unsigned int bytesDecoded = 0;
    AsnTag tagId0;
    AsnLen elmtLen0;
    DataContainerObjectChoice *dataObject;
    AuthenticationObjectChoice *authenticationObject;
    AuthenticationObject_PasswordAttributes* pwd;
    AccessControlRule *accessControlRule;
    SecurityCondition *securityCondition, *securityConditionAuthID;
    AsnOcts *authId1;

    if(setjmp(env)!=0) exit(0);

    /*
    ** Retrieve the access rule for reading from the Data Object
    */
    cioDO = (CIOChoice *)calloc(1, sizeof(DataContainerObjectChoice));

    SBufInstallData(&dob, DOBER, DOBERLength);
    SBufToGenBuf(&dob, &dogb);

    tagId0 = BDecTag(dogb, &bytesDecoded, env);
    elmtLen0 = BDecLen(dogb, &bytesDecoded, env);

    BDecCIOChoiceContent(dogb, tagId0, elmtLen0, cioDO, &bytesDecoded, env);

```

```

dataObject = (DataContainerObjectChoice *) (cioDO->a.dataContainerObjects->a.objects->first->data);

FOR_EACH_LIST_ELMT(accessControlRule, dataObject->a.iso7816DO->commonObjectAttributes-
>accessControlRules)
{
    if(accessControlRule->accessMode.bits && READ_FLAG)
        break;
}

if(accessControlRule == NULL)
    exit(0);

securityCondition = accessControlRule->securityCondition;

if(securityCondition->choiceId != SECURITYCONDITION_SEACOS_OR)
    exit(0);

securityConditionAuthID = (SecurityCondition *) (securityCondition->a.seacos_or->first->data);

if(securityConditionAuthID->choiceId != SECURITYCONDITION_AUTHID)
    exit(0);

authId1 = (AsnOcts *) securityConditionAuthID->a.authId;

/*
** Find the Authentication Object associated with the first term in the OR
*/
cioAO = (CIOChoice *) calloc(1, sizeof(AuthenticationObjectChoice));

SBufInstallData(&aob, AOBER, AOBERLength);
SBufToGenBuf(&aob, &aogb);

tagId0 = BDecTag(aogb, &bytesDecoded, env);
elmtLen0 = BDecLen(aogb, &bytesDecoded, env);

BDecCIOChoiceContent(aogb, tagId0, elmtLen0, cioAO, &bytesDecoded, env);

FOR_EACH_LIST_ELMT(authenticationObject, cioAO->a.dataContainerObjects->a.objects)
{
    if(authenticationObject->choiceId == AUTHENTICATIONOBJECTCHOICE_PWD)
    {
        pwd = authenticationObject->a.pwd;
        if((authId1->octetLen == pwd->commonObjectAttributes->label.octetLen) &&
            memcmp(authId1->octets, pwd->commonObjectAttributes->label.octets, authId1->octetLen) == 0)
        {
            /*
            ** Found the AO that goes with the first term in the OR
            ** condition associated with the READ access mode of the DO
            */
            break;
        }
    }
}
}
}

```

E.3.4 BER Encoding

```
<EF_DO>
0xa7,0x46,0xa0,0x44,0xa0,0x42,0x30,0x34,0x0c,0x04,0x44,0x4f,0x2d,0x31,0x03,
0x02,0x06,0x40,0x30,0x28,0x30,0x12,0x03,0x02,0x04,0x80,0xa2,0x0c,0x04,0x04,
0x41,0x4f,0x2d,0x31,0x04,0x04,0x41,0x4f,0x2d,0x32,0x30,0x12,0x03,0x02,0x04,
0x40,0xa1,0x0c,0x04,0x04,0x41,0x4f,0x2d,0x31,0x04,0x04,0x41,0x4f,0x2d,0x32,
0x30,0x00,0xa1,0x08,0xa0,0x06,0x60,0x04,0x80,0x02,0x01,0x02
</EF_DO>

<EF_AO>
0xa8,0x3e,0xa0,0x3c,0x30,0x3a,0x30,0x0a,0x0c,0x04,0x41,0x4f,0x2d,0x31,0x03,0x02,
0x06,0x40,0x30,0x0c,0x04,0x04,0x41,0x4f,0x2d,0x31,0x02,0x01,0x01,0x80,0x01,0x02,
0xa1,0x1e,0x30,0x1c,0x03,0x03,0x04,0x04,0x00,0x0a,0x01,0x01,0x02,0x01,0x04,0x02,
0x01,0x0c,0x02,0x01,0x08,0x04,0x01,0xff,0x30,0x06,0x04,0x04,0x3f,0x00,0x40,0x45
</EF_AO>

<EF_SK>
0xa3,0x3b,0xa0,0x39,0xaf,0x37,0x30,0x10,0x0c,0x04,0x53,0x4b,0x2d,0x31,0x03,0x02,
0x06,0x40,0x04,0x04,0x41,0x4f,0x2d,0x31,0x30,0x14,0x04,0x04,0x53,0x4b,0x2d,0x31,
0x03,0x02,0x02,0x00,0x01,0x01,0xff,0x03,0x02,0x04,0x10,0x02,0x01,0x0a,0xa0,0x05,
0x30,0x03,0x02,0x01,0x40,0xa1,0x06,0x30,0x04,0x06,0x01,0x58,0x58
</EF_SK>
```

E.4 Encoding of a Certificate

E.4.1 Cryptographic Information Application Example Description

A description of an X.509 certificate.

E.4.2 ASN.1 Encoding of an X.509 Certificate

```
certificates objects { -- SEQUENCE OF --
  x509Certificate { -- SEQUENCE --
    commonObjectAttributes { -- SEQUENCE --
      label '43657274696669636174652031'H -- "Certificate 1" --,
      flags '40'H,
      authId '17'H,
      userConsent 5
    },
    classAttributes { -- SEQUENCE --
      id '41444d'H -- "ADM" --,
      authority FALSE,
      identifier { -- SEQUENCE --
        idType 0,
        idValue '3132333435363738'H -- "12345678" --
      },
      certHash { -- SEQUENCE --
        hashAlg { -- SEQUENCE --
          algorithm {0 17 34 51},
          parameters '332211'H
        },
        certId { -- SEQUENCE --
          issuer iPAddress 'c0a82d01'H,
          serialNumber 13107
        },
        hashVal '998877'H
      },
    },
  },
}
```

```

trustedUsage { -- SEQUENCE --
    keyUsage '2000'H
},
identifiers { -- SEQUENCE OF --
    { -- SEQUENCE --
        idType 5,
        idValue '616263'H -- "abc" --
    },
    { -- SEQUENCE --
        idType 5,
        idValue '78797a'H -- "xyz" --
    }
},
typeAttributes { -- SEQUENCE --
    value indirect path { -- SEQUENCE --
        efidOrPath '3f004042'H -- "? @B" --
    },
    subject rdnSequence { -- SEQUENCE OF --
        { -- SET OF --
            { -- SEQUENCE --
                type {1 11 68},
                value NULL,
                valuesWithContext { -- SET OF --
                    { -- SEQUENCE --
                        distingAttrValue '5577'H,
                        contextList { -- SET OF --
                            { -- SEQUENCE --
                                contextType {2 40 86},
                                contextValues { -- SET OF --
                                    '876543'H
                                },
                                fallback TRUE
                            }
                        }
                    }
                }
            }
        }
    },
    issuer rdnSequence { -- SEQUENCE OF --
        { -- SET OF --
            { -- SEQUENCE --
                type {2 5 102},
                value NULL,
                valuesWithContext { -- SET OF --
                    { -- SEQUENCE --
                        distingAttrValue '8899'H,
                        contextList { -- SET OF --
                            { -- SEQUENCE --
                                contextType {2 40 86},
                                contextValues { -- SET OF --
                                    '785634'H
                                },
                                fallback TRUE
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    },
    serialNumber 22376
  }
}
}

```

E.4.3 Code from the ASN.1 for Encoding and Decoding BER

```

/*
** Example of Code for Encoding the BER
*/
void X509Certificate(unsigned char *label,
                    unsigned char objectFlags,
                    unsigned char *iD, unsigned int iDLength,
                    unsigned char authority,
                    unsigned short usageFlags,
                    unsigned int externalIdentifierType,
                    unsigned char *externalIdentifier,
                    unsigned char *path, unsigned int pathLength,
                    unsigned char *BER, unsigned int *BERLength) // Output
{
  unsigned int l;
  SBuf b;
  GenBuf *gb;
  unsigned char buffer[1024];

  CIOChoice *cio;
  CertificateChoice *prk, **prkp;
  AccessControlRule *acr, **acrp;
  SecurityCondition *sc, **scp;
  CredentialIdentifier *cid, **cidp;
  RelativeDistinguishedName *rdn, **rdnp;
  AttributeTypeAndDistinguishedValue *atadv, **atadvp;
  AttributeTypeAndDistinguishedValueSetOfSeq *atadvsos, **atadvssosp;
  Context *atadvso, **atadvso;
  AsnAny *any, **anyp;

  SecurityCondition securityCondition1;
  SecurityCondition securityCondition2;
  AsnOcts authId11 = { sizeof(AuthId11), AuthId11 };
  AsnOcts authId12 = { sizeof(AuthId12), AuthId12 };
  AsnOcts authId21 = { sizeof(AuthId21), AuthId21 };
  AsnOcts authId22 = { sizeof(AuthId22), AuthId22 };

  CertificateObject_X509CertificateAttributes pattn = { 0 };
  CommonObjectAttributes commonObjAttr = { 0 };
  CommonCertificateAttributes commonCertificateAttr = { 0 };
  X509CertificateAttributes x509CertificateAttr = { 0 };

  CredentialIdentifier credentialIdentifier = { 0 };
  Path pathOctets = { 0 };
  AsnOcts issuerHash = { 0 };
  AsnOcts issuerHash1 = { sizeof(identifier1), identifier1 };
  AsnOcts issuerHash2 = { sizeof(identifier2), identifier2 };

  AsnOcts asnATADVvalue = { sizeof(ATADVvalue), ATADVvalue };
  AsnOcts asnATADVdistvalue = { sizeof(ATADVdistvalue), ATADVdistvalue };
  AsnOcts asnATADVvalueIssuer = { sizeof(ATADVvalueIssuer), ATADVvalueIssuer };

```

```

AsnOcts asnATADVdistvalueIssuer = { sizeof(ATADVdistvalueIssuer ), ATADVdistvalueIssuer };

char commonObjectFlags[1]                = { 0 };
AsnBits commonFlagsAsnBits                = { 2, commonObjectFlags };

char accessControlRuleFlags1[1]           = { 0 };
AsnBits accessControlRuleAsnBits1         = { 4, accessControlRuleFlags1 };
char accessControlRuleFlags2[1]           = { 0 };
AsnBits accessControlRuleAsnBits2         = { 4, accessControlRuleFlags2 };

char usageFlagBits[2]                     = { 0 };
AsnBits usageFlagsAsnBits                 = { 10, usageFlagBits };

CertHash certHash = { 0 };
AlgorithmIdentifier algoId = { 0 };
AsnOcts parameters = { 0 };

CertId certId = { 0 };
GeneralName issuer;

Usage usage = { 0 };

AsnAny contextValue1;
AsnOcts contextValue1Octs = { sizeof(ContextValue1Octs), ContextValue1Octs };
AsnAny contextValue1Issuer;
AsnOcts contextValue1OctsIssuer = { sizeof(ContextValue1OctsIssuer), ContextValue1OctsIssuer };

InitAnyCryptographicInformationFramework();
InitAnyInformationFramework2();

SBufInit(&b, buffer, sizeof(buffer));
SBufResetInWriteRvsMode(&b);
SBufToGenBuf(&b, &gb);

/*
** Section 8.3 The CIOChoice type
**
** "EF.OD shall contain the concatenation of 0, 1, or more DER-encoded CIOChoice values."
**
**
*/
cio = (CIOChoice *)calloc(1, sizeof(CertificateChoice));
cio->choiceId = CIOCHOICE_CERTIFICATES;

/*
** "It is expected that an EF.OD entry will usually reference a separate file (the path
** choice of PathOrObjects) containing CIOs of the indicated type. An entry may, however,
** hold CIOs directly (the objects choice of PathOrObjects), if the objects and the EF.OD
** file have the same access control requirements."
**
** PathOrObjects{CertificateChoice}
*/
cio->a.certificates = (P15Certificates *)calloc(1, sizeof(P15Certificates));
cio->a.certificates->choiceId = PATHOROBJECTS_CERTIFICATECHOICE_OBJECTS;
cio->a.certificates->a.objects = AsnListNew(sizeof (void*));

/*
** Section 8.4.1 CertificateChoice
**
** "This type contains information pertaining to a private key. Each value

```



```

** consists of attributes common to any object, any key, any private key,
** and attributes particular to the key."
*/
prkp = (CertificateChoice **)AsnListAppend(cio->a.certificates->a.objects);

*prkp = prk = calloc(1, sizeof(CertificateChoice));

prk->choiceId = CERTIFICATECHOICE_X509CERTIFICATE;

prk->a.x509Certificate = &pattr;

pattr.commonObjectAttributes = &commonObjAttr;
pattr.classAttributes        = &commonCertificateAttr;
pattr.subClassAttributes     = NULL;
pattr.typeAttributes         = &x509CertificateAttr;

/*
** Section 8.2.8 CommonObjectAttributes
**
** "This type is a container for attributes common to all CIOs."
**
*/
commonObjAttr.label.octs = label;
commonObjAttr.label.octetLen = strlen(label);

commonObjectFlags[0] = objectFlags;
commonObjAttr.flags = commonFlagsAsnBits;

commonObjAttr.authId.octetLen = sizeof(authId);
commonObjAttr.authId.octs = authId;

commonObjAttr.userConsent = &one;

/*
** Section 8.2.15 CommonCertificateAttributes
**
** "When a public key in a certificate referenced by a certificate
** information object corresponds to a private key referenced by a
** private key information object, then the information objects
** shall share the same value for the iD field. This requirement
** will simplify searches for a private key corresponding to a
** particular certificate and vice versa. Multiple certificates
** for the same key shall share the same value for the iD."
*/
commonCertificateAttr.iD.octetLen = iDLength;
commonCertificateAttr.iD.octs = iD;
commonCertificateAttr.authority = &authority;

issuerHash.octs = externalIdentifier;
issuerHash.octetLen = strlen(externalIdentifier);
credentialIdentifier.idValue.value = &issuerHash;
SetAnyTypeByInt(&(credentialIdentifier.idValue), externalIdentifierType);

commonCertificateAttr.identifier = &credentialIdentifier;

/* Cert Hash */
commonCertificateAttr.certHash = &certHash;
certHash.hashAlg = &algoId;
algoId.algorithm.octetLen = sizeof(algoId);

```

```

algoId.algorithm.octs = algo1Id;

parameters.octetLen = sizeof(algo1Pm);
parameters.octs = algo1Pm;
algoId.parameters.value = &parameters;
SetAnyTypeByInt(&algoId.parameters, externalIdentifierType);

/* Cert Identifier */
certHash.certId = &certId;
certId.issuer = &issuer;
issuer.choiceId = GENERALNAME_IPADDRESS;
issuer.a.iPAddress = (AsnOcts *)calloc(1, sizeof(AsnOcts));
issuer.a.iPAddress->octetLen = sizeof(issuerIPAddress);
issuer.a.iPAddress->octs = issuerIPAddress;
certId.serialNumber = 0x3333;

/* Cert Hash */
certHash.hashVal.bitLen = 8*sizeof(hashbits);
certHash.hashVal.bits = hashbits;

/* Usage */
commonCertificateAttr.trustedUsage = &usage;
usageFlagBits[0] = (unsigned char)(usageFlags>>8);
usageFlagBits[1] = (unsigned char)(usageFlags);
usage.keyUsage = usageFlagsAsnBits;

/* Identifiers */
commonCertificateAttr.identifiers = AsnListNew(sizeof(void*));
cidp = (CredentialIdentifier **)AsnListAppend(commonCertificateAttr.identifiers);
*cidp = cid = calloc(1, sizeof(CredentialIdentifier));
cid->idType = externalIdentifierType;
cid->idValue.value = &issuerHash1;
SetAnyTypeByInt(&(cid->idValue), externalIdentifierType);

cidp = (CredentialIdentifier **)AsnListAppend(commonCertificateAttr.identifiers);

*cidp = cid = calloc(1, sizeof(CredentialIdentifier));

cid->idType = externalIdentifierType;
cid->idValue.value = &issuerHash2;
SetAnyTypeByInt(&(cid->idValue), externalIdentifierType);

/*
** Section 8.7.2 X509 Certificate Attributes
**
** "X509CertificateAttributes.value: The value shall be a ReferencedValue
** either identifying a file containing a DER encoded certificate at the
** given location, or a URL pointing to some location where the certificate can be found.
**
** "X509CertificateAttributes.subject, X509CertificateAttributes.issuer and
** X509CertificateAttributes.serialNumber: The semantics of these fields is the
** same as for the corresponding fields in ISO/IEC 9594-8:1998. The values of these
** fields shall be exactly the same as for the corresponding fields in the certificate itself.
** The reason for making them optional is to provide some space-efficiency, since they already
** are present in the certificate itself."
*/
x509CertificateAttr.value = (ObjectValue *)calloc(1, sizeof(ObjectValue));

x509CertificateAttr.value->choiceId = OBJECTVALUE_INDIRECT;
x509CertificateAttr.value->a.indirect = (ReferencedValue *)calloc(1, sizeof(ReferencedValue));

```

```

x509CertificateAttr.value->a.indirect->choiceId = REFERENCEDVALUE_PATH;

pathOctets.efidOrPath.octs = (char *)calloc(1, pathLength);
memcpy(pathOctets.efidOrPath.octs, path, pathLength);
pathOctets.efidOrPath.octetLen = pathLength;
x509CertificateAttr.value->a.indirect->a.path = &pathOctets;
/* Subject */
x509CertificateAttr.subject = (Name *)calloc(1, sizeof(Name));
x509CertificateAttr.subject->choiceId = NAME_RDNSEQUENCE;
x509CertificateAttr.subject->a.rdnSequence = AsnListNew(sizeof (void*));
rdnp = (RelativeDistinguishedName **)AsnListAppend(x509CertificateAttr.subject->a.rdnSequence);
*rdnp = rdn = AsnListNew(sizeof (void*));
atadvp = (AttributeTypeAndDistinguishedValue **)AsnListAppend(rdn);
*atadvp=atadv=
    (AttributeTypeAndDistinguishedValue*)
        calloc(1, sizeof(AttributeTypeAndDistinguishedValue));
atadv->type.octetLen = sizeof(ATADVtype);
atadv->type.octs = ATADVtype;

atadv->value.value = &asnATADVvalue;
SetAnyTypeByOid(&(atadv->value), &noRevAvail);

atadv->valuesWithContext = AsnListNew(sizeof (void*));
atadvssosp = (AttributeTypeAndDistinguishedValueSetOfSeq **)AsnListAppend(atadv->valuesWithContext);
*atadvssosp=atadvsos=
    (AttributeTypeAndDistinguishedValueSetOfSeq *)
        calloc(1, sizeof(AttributeTypeAndDistinguishedValueSetOfSeq));

atadvsos->distingAttrValue.value = &asnATADVdistvalue;
SetAnyTypeByInt(&(atadvsos->distingAttrValue), externalIdentifierType);

atadvsos->contextList = AsnListNew(sizeof (void*));
atadvssossop = (Context **)AsnListAppend(atadvsos->contextList);
*atadvssossop = atadvosso = (Context *)calloc(1, sizeof(Context));
atadvosso->contextType.octetLen = sizeof(contextTypeIssuer);
atadvosso->contextType.octs = contextTypeIssuer;
atadvosso->contextValues = AsnListNew(sizeof (void*));
anyp = (AsnAny **)AsnListAppend(atadvosso->contextValues);
contextValue1.value = &contextValue1Octs;
SetAnyTypeByInt(&(contextValue1), externalIdentifierType);
*anyp = any = &contextValue1;

atadvosso->fallback = &True;

atadv->primaryDistinguished = FALSE;

/* Issuer */
x509CertificateAttr.issuer = (Name *)calloc(1, sizeof(Name));
x509CertificateAttr.issuer->choiceId = NAME_RDNSEQUENCE;
x509CertificateAttr.issuer->a.rdnSequence = AsnListNew(sizeof (void*));
rdnp = (RelativeDistinguishedName **)AsnListAppend(x509CertificateAttr.issuer->a.rdnSequence);
*rdnp = rdn = AsnListNew(sizeof (void*));
atadvp = (AttributeTypeAndDistinguishedValue **)AsnListAppend(rdn);
*atadvp=atadv=
    (AttributeTypeAndDistinguishedValue *)
        calloc(1, sizeof(AttributeTypeAndDistinguishedValue));
atadv->type.octetLen = sizeof(ATADVtypeIssuer);
atadv->type.octs = ATADVtypeIssuer;

atadv->value.value = &asnATADVvalueIssuer;

```

```

SetAnyTypeByOid(&(atadv->value), &noRevAvail);

atadv->valuesWithContext = AsnListNew(sizeof (void*));
atadvssosp= (AttributeTypeAndDistinguishedValueSetOfSeq **)AsnListAppend(atadv->valuesWithContext);
*atadvssosp=atadvssos=
    (AttributeTypeAndDistinguishedValueSetOfSeq *)
        calloc(1, sizeof(AttributeTypeAndDistinguishedValueSetOfSeq));
atadvssos->distingAttrValue.value = &asnATADVdistvalueIssuer;
SetAnyTypeByInt(&(atadvssos->distingAttrValue), externalIdentifierType);

atadvssos->contextList = AsnListNew(sizeof (void*));
atadvssosop = (Context **)AsnListAppend(atadvssos->contextList);
*atadvssosop = atadvssosso = (Context *)calloc(1, sizeof(Context));
atadvssosso->contextType.octetLen = sizeof(contextTypeIssuer);
atadvssosso->contextType.octs = contextTypeIssuer;
atadvssosso->contextValues = AsnListNew(sizeof (void*));
anyp = (AsnAny **)AsnListAppend(atadvssosso->contextValues);
contextValue1Issuer.value = &contextValue1OctsIssuer;
SetAnyTypeByInt(&(contextValue1Issuer), externalIdentifierType);
*anyp = any = &contextValue1Issuer;

atadvssosso->fallback = &True;

atadv->primaryDistinguished = FALSE;

x509CertificateAttr.serialNumber = &certSerialNumber;

/*
** Print the Certificate Data Object
*/
PrintCIOChoice(stdout, cio, 3);

/*
** BER Encode the Certificate Data Object
*/
*BERLength = BEncCIOChoiceContent(gb,cio);
GenBufResetInReadMode(gb);

l = 0;
memcpy(BER, GenBufGetSeg(gb, &l), *BERLength);
}

/*
** Example of Code for Decoding the BER
*/
Path_to_X509_Certificate(unsigned char *BER, unsigned int BERLength)
{
    SBuf b;
    GenBuf *gb;
    unsigned int bytesDecoded = 0;
    ENV_TYPE env;
    AsnTag tagId0;
    AsnLen elmtLen0;
    CIOChoice *cio;
    CertificateChoice *certificate;
    CertificateObject_X509CertificateAttributes* x509Certificate;
    X509CertificateAttributes* typeAttributes;
    ObjectValue* value;
    unsigned int i, pathLength;
    unsigned char *path;

```

```

if(setjmp(env) != 0) exit(0);

cio = calloc(1, sizeof(CIOChoice));

SBufInstallData(&b, BER, BERLength);
SBufToGenBuf(&b, &gb);
tagId0 = BDecTag(gb, &bytesDecoded, env);
elmtLen0 = BDecLen(gb, &bytesDecoded, env);

/*
** Decode the X.509 Certificate
*/
BDecCIOChoiceContent(gb, tagId0, elmtLen0, cio, &bytesDecoded, env);

/*
** Find the path to the certificate
*/
certificate = (CertificateChoice *) (cio->a.certificates->a.objects->first->data);

x509Certificate = certificate->a.x509Certificate;

typeAttributes = x509Certificate->typeAttributes;

value = typeAttributes->value;

printf("Path to Certificate: ");
pathLength = value->a.indirect->a.path->efidOrPath.octetLen;
path = value->a.indirect->a.path->efidOrPath.octs;
for(i = 0; i < pathLength; i+=2)
    printf("0x%02x%02x ", path[i], path[i+1]);
printf("\n");
}

```

E.4.4 BER Encoding

```

<BER>
0xa4,0x81,0xe1,0xa0,0x81,0xde,0x30,0x81,0xdb,0x30,0x19,0x0c,0x0d,0x43,0x65,0x72,
0x74,0x69,0x66,0x69,0x63,0x61,0x74,0x65,0x20,0x31,0x03,0x02,0x06,0x40,0x04,0x01,
0x17,0x02,0x01,0x05,0x30,0x58,0x04,0x03,0x41,0x44,0x4d,0x01,0x01,0x00,0x30,0x0d,
0x02,0x01,0x00,0x60,0x08,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0xa0,0x22,0xa0,
0x0c,0x30,0x0a,0x06,0x03,0x11,0x22,0x33,0x60,0x03,0x33,0x22,0x11,0xa1,0x0c,0x30,
0x0a,0x87,0x04,0xc0,0xa8,0x2d,0x01,0x02,0x02,0x33,0x33,0x03,0x04,0x00,0x99,0x88,
0x77,0xa1,0x05,0x03,0x03,0x06,0x20,0x00,0xa2,0x14,0x30,0x08,0x02,0x01,0x05,0x60,
0x03,0x61,0x62,0x63,0x30,0x08,0x02,0x01,0x05,0x60,0x03,0x78,0x79,0x7a,0xa1,0x64,
0x30,0x62,0x30,0x06,0x04,0x04,0x3f,0x00,0x40,0x42,0x30,0x28,0x31,0x26,0x30,0x24,
0x06,0x02,0x33,0x44,0x60,0x02,0x44,0x33,0x31,0x1a,0x30,0x18,0xa0,0x04,0x60,0x02,
0x55,0x77,0x31,0x10,0x30,0x0e,0x06,0x02,0x78,0x56,0x31,0x05,0x60,0x03,0x87,0x65,
0x43,0x01,0x01,0xff,0xa0,0x2a,0x30,0x28,0x31,0x26,0x30,0x24,0x06,0x02,0x55,0x66,
0x60,0x02,0x66,0x77,0x31,0x1a,0x30,0x18,0xa0,0x04,0x60,0x02,0x88,0x99,0x31,0x10,
0x30,0x0e,0x06,0x02,0x78,0x56,0x31,0x05,0x60,0x03,0x78,0x56,0x34,0x01,0x01,0xff,
0x02,0x02,0x57,0x68
</BER>

```

Table E.2 is a diagrammatic representation of the BER encoding.

Table E.2 — EF.CD of X.509 Certificate

			Data Type		
A4	81	CIOChoice: Certificate			
	E1				
A0	81	Certificate Choice: X.509 certificate			
	DE				
	30	X.509 certificate Object			
	DB				
		30	19	Common Object Attributes	
		0C	0D	Label	43, 65, 72, 74, 69, 66, 69, 63, 61, 74, 65, 20, 31
		03	02	Flags	05 40
		04	01	authId	17
		02	01	userConsent	05
		30	58	Common Certificate Attributes	
		04	03	iD	41, 44, 4D
		01	01	Authority	00
		30	0D	Identifier	
		02	01	idType	00
		60	08	idValue	31, 32, 33, 34, 35, 36, 37, 38
A0	22	certHash			

Table E.2 (continued)

A0	0C	hashAlg							
		30	0A						
				06	03	Algorithm	11, 22, 33	OID	
				60	03	parameters	33, 22, 11	OpenType	
A1	0C	certID							
		30	0A						
				87	04	Issuer(iPAddress)	C0, A8, 2D, 01	OCTET STRING	
				02	02	serialNumber	33, 33	INTEGER	
03	04	hashVal							INTEGER
A1	05	trustUsage							
03	03	keyUsage					06, 20, 00	BIT STRING	
A2	14	identifiers							
30	08	CredentialIdentifier							
		02	01	idType			05	INTEGER	
		60	03	idValue			61, 62, 63	OpenType	
30	08	CredentialIdentifier							
		02	01	idType			05	INTEGER	
		60	03	idValue			78, 79, 7A	OpenType	

Table E.2 (continued)

A1	64	X509CertificateAttributes									
		30	62	Choice							
		30	06	Value							
		04	04	efidOrPath				3F, 00, 40, 42		OCTET STRING	
		30	28	subject (RDNSequence)							
		34	26	rdnSequence							
		30	24	AttributeTypeAndDispinguishedValue							
		06	02	Type		33, 44		OID			
		60	02	Value		44, 33		OpenType			
		31	1A	SET valuesWithContext							
		30	18	SEQUENCE							
		A0	04	SupportedAttributes							
		60	02	55, 77				OpenType			
		31	10	contextList							
		30	0E	SEQUENCE							
		06	02	contextType				OID			
		31	05	SET							
		60	03	Context		87, 65, 43		OpenType			
		01	01	fallback		FF		BOOLEAN			

Table E.2 (continued)

A0		2A		issuer (RDNSequence)													
		30		28		Choice :											
		31		26		SET of AttributeTypeAndValue											
		30		24													
		06		02		Type		55, 66		OID							
		60		02		Value		66, 77		OpenType							
		31		1A		SET valuesWithContext											
		30		18		SEQUENCE											
		A0		04		SupportedAttributes											
		60		02		88, 99		OpenType									
		31		10		SET contextList											
		30		0E		SEQUENCE											
		06		02		contextType		78, 56		OID							
		31		05		SET Cotext											
		60		03		Context		78, 56, 34									
		01		01		fallback		FF		BOOLEAN							
02		02		serial Number		57, 68				INTEGER							

E.5 Encoding of the ESIGN Cryptographic Information Application

E.5.1 Cryptographic Information Application Example Description

The encoding is an example of a Cryptographic Information Application for the ESIGN as described in Section 16 of CWA 14890-1:2004

E.5.2 ASN.1 Encoding of the IAS Cryptographic Information Application

```
{ -- SEQUENCE OF --
  privateKeys path { -- SEQUENCE --
    efidOrPath '4001'H
  },
  publicKeys path { -- SEQUENCE --
    efidOrPath '4002'H
  },
  certificates path { -- SEQUENCE --
    efidOrPath '4005'H
  },
  trustedCertificates path { -- SEQUENCE --
    efidOrPath '4004'H
  },
  dataContainerObjects path { -- SEQUENCE --
    efidOrPath '4006'H
  },
  authObjects path { -- SEQUENCE --
    efidOrPath '4003'H
  }
}

cardInfo { -- SEQUENCE --
  version 2,
  serialNumber '0102030405060708'H,
  manufacturerID '41434d45'H -- "ACME" --,
  label '5369676e617475726520417070666963617469666e'H -- "Signature Application" --,
  cardflags '60'H,
  seInfo { -- SEQUENCE OF --
    { -- SEQUENCE --
      se 1,
      aid 'a000000167455349474e'H
    },
    { -- SEQUENCE --
      se 2,
      aid 'a000000167455349474e'H
    }
  },
  supportedAlgorithms { -- SEQUENCE OF --
    { -- SEQUENCE --
      reference 1,
      algorithm 544,
      parameters 'H -- "" --,
      supportedOperations '02'H,
      objId {0 1 3 14 3 2 26},
      algRef 16
    },
    { -- SEQUENCE --
      reference 2,
      algorithm -2147483648,
      parameters 'H -- "" --,
```

```

        supportedOperations '40'H,
        objId {0 1 3 36 3 4 3 2 1},
        algRef 17
    },
    { -- SEQUENCE --
        reference 3,
        algorithm 544,
        parameters 'H -- "" --,
        supportedOperations '40'H,
        objId {0 1 2 72 113 37 1 1 5},
        algRef 18
    },
    { -- SEQUENCE --
        reference 4,
        algorithm -2147483647,
        parameters 'H -- "" --,
        supportedOperations '50'H,
        objId {0 1 3 36 7 2 1 1},
        algRef 23
    },
    { -- SEQUENCE --
        reference 5,
        algorithm -2147483646,
        parameters 'H -- "" --,
        supportedOperations '10'H,
        objId {0 1 3 36 3 4 3 2 1}
    }
},
issuerId '4d61696e205374726565742042616e6b'H -- "Main Street Bank" --,
holderId '53616c6c7920477265656e'H -- "Sally Green" --,
lastUpdate generalizedTime '31393835313130363231303632372e335a'H -- "19851106210627.3Z" --,
preferredLanguage '4573706572616e746f'H -- "Esperanto" --
}

AuthenticationObjects { -- SEQUENCE OF --
    pwd { -- SEQUENCE --
        commonObjectAttributes { -- SEQUENCE --
            label '476c6f62616c2050617373776f7264'H -- "Global Password" --,
            flags '40'H,
            authId '03'H,
            accessControlRules { -- SEQUENCE OF --
                { -- SEQUENCE --
                    accessMode '20'H,
                    securityCondition authReference { -- SEQUENCE --
                        authMethod 'c0'H,
                        seIdentifier 2
                    }
                }
            },
            classAttributes { -- SEQUENCE --
                authId '01'H,
                authReference 1,
                seIdentifier 2
            },
            typeAttributes { -- SEQUENCE --
                pwdFlags '08'H,
                pwdType 0,
                minLength 4,
                storedLength 0,

```