ANÁLISIS DE LA ESTRUCTURA INTERNA DEL DNI ELECTRÓNICO

Autor: Javier Vicente Vallejo
Url: http://www.vallejo.cc

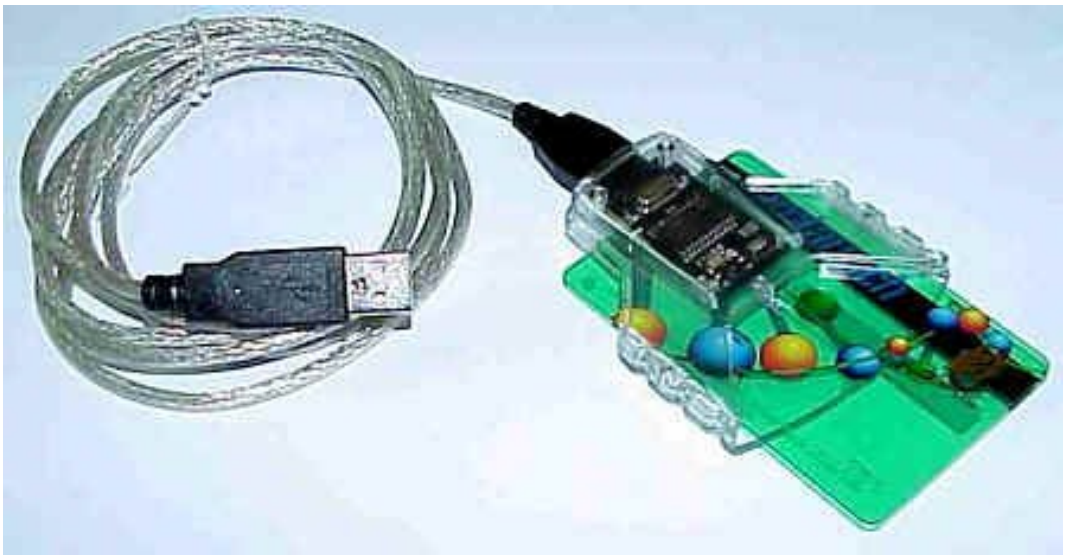
0. ÍNDICE

1. INTRODUCCIÓN

Hace unos días fuí a renovar mi DNI. Me dieron uno electrónico y me ofrecieron un lector para operar en internet a cambio de recibir una charla. No dejó de sorprenderme lo mucho que insistía la chica de la charla en que el DNI, internamente, solamente contenía los mismos datos que se pueden leer en el soporte físico, más dos claves públicas, la de autenticación y la de firmado de documentos. Insistió tanto que hasta resultaba sospechoso. Su insistencia apoyada por mi paranoia y desconfianza me impulsaron a ponerme a enredar un poco con el DNI y el lector, a ver se podía sacar, y fruto de ello este artículo en el que comparto con vosotros lo que he podido ver y en el que resumo lo que he aprendido leyendo sobre este tema.

2. EL LECTOR

El lector es de la marca GEMALTO, modelo PC TWIN.



Junto con el lector se distribuye un cd con los drivers, documentación, certificados, software, apis para desarrollo e incluso código fuente.

Aplicaciones:

e:\DGPolicia\--- DESCARGAS ---\WINDOWS\DNIe v4_0_0.exe → navegadores
e:\DGPolicia\--- DESCARGAS ---\WINDOWS\SOFTWARE PAD VIRTUAL DNIE → cambio de pin

Certificados:

e:\DGPolicia\AUTORIDAD_CERTIFICACION\RAIZ\ACRAIZ-SHA1.zip
e:\DGPolicia\AUTORIDAD_CERTIFICACION\RAIZ\ACRAIZ-SHA2.zip
e:\DGPolicia\AUTORIDAD_CERTIFICACION\SUBORDINADA\ACDNIE001-SHA1.zip
e:\DGPolicia\AUTORIDAD_CERTIFICACION\SUBORDINADA\ACDNIE001-SHA2.zip
e:\DGPolicia\AUTORIDAD_CERTIFICACION\SUBORDINADA\ACDNIE002-SHA1.zip
e:\DGPolicia\AUTORIDAD_CERTIFICACION\SUBORDINADA\ACDNIE002-SHA2.zip
e:\DGPolicia\AUTORIDAD_CERTIFICACION\SUBORDINADA\ACDNIE003-SHA1.zip
e:\DGPolicia\AUTORIDAD_CERTIFICACION\SUBORDINADA\ACDNIE003-SHA2.zip
e:\DGPolicia\AUTORIDAD_VALIDACION\FNMT\AVDNIEFNMTSHA1.zip
e:\DGPolicia\AUTORIDAD_VALIDACION\FNMT\AVDNIEFNMTSHA2.zip
e:\DGPolicia\AUTORIDAD_VALIDACION\MAP\AVDNIEMAPSHA1.zip
e:\DGPolicia\AUTORIDAD_VALIDACION\MAP\AVDNIEMAPSHA2.zip
e:\DGPolicia\AUTORIDAD_VALIDACION\MITYC\AVDNIEMITYCSHA1.zip
e:\DGPolicia\AUTORIDAD_VALIDACION\MITYC\AVDNIEMITYCSHA2.zip

Estandares:

e:\DGPolicia\ESTANDARES\cwa14167-01-2003-Jun.pdf
e:\DGPolicia\ESTANDARES\cwa14172-04-2004-Mar.pdf
e:\DGPolicia\ESTANDARES\cwa14890-02-2004-May.pdf

Apis:

e:\HARDWARE\GEMALTO_SCR\DOCS\CILK_API.zip
e:\HARDWARE\GEMALTO_SCR\DOCS\CTAPI.zip
e:\HARDWARE\GEMALTO_SCR\DOCS\GEM_PCTWIN.zip
e:\HARDWARE\GEMALTO_SCR\DOCS\Synchronous_card_API_011.zip
e:\HARDWARE\GEMALTO_SCR\DRIVERS\WINDOWS_XP\GemCCID_64en-us.msi

Aplicación de testeo del lector:

e:\TEST

Instalamos los drivers, aplicaciones, etc... Entre las aplicaciones tenemos una hecha en java para el cambio de pin del DNIE y otra (una dll) que se integra con los navegadores para las gestiones en webs (c:\WINDOWS\system32\UsrDNIeCertStore.dll). En e:\test hay una aplicación para testear el funcionamiento correcto del lector.

3. LA SMARTCARD

Modelo del Chip: **st19wl34 y ICC ST19wl34**.
Sistema operativo: DNIe v1.1.
Capacidad: 34Kbytes Eeprom.

(http://www.dnielectronico.es/seccion_integradores/espec_uno.html).

4. EL SNIFFER USB

Necesitaremos un sniffer usb para ver el intercambio de APDUs entre la smartcard y las aplicaciones. Yo he usado **UsbSnoop**: http://sourceforge.net/projects/usbsnoop/

Los resultados de usbsnoop están llenos de paja relacionada con la gestión del puerto usb que no nos interesan. El pequeño script que viene a continuación nos limpia un poco los resultados de usbsnoop y nos quedamos practicamente con lo que toca al protocolo de comunicación con las smartcards (especificado en ISO 7816).

```
import sys
def hex2ascii(s):
	l=s.split(" ")[5:]
	l2=[]
	rs=""
	for e in l:
		#if (int(e,16)>=0x61 and int(e,16)<0x7a) or (int(e,16)>=0x41 and int(e,16)<0x5a):
		if (int(e,16)>=0x20 and int(e,16)<0x7e):
			rs+=chr(int(e,16))
		else:
			rs+="?"
	return rs
f=open(sys.argv[1])
f2=open(sys.argv[1]+".simple.log","wb")
s=f.readline()
lastdir=""
while len(s):
	if len(s) >= len("    0000") and s[0:len("    0000")]=="    0000" and ":" in s:
		if len(s) >= len("    00000000:") and s[0:len("    00000000:")]=="    00000000:":
			f2.write(lastdir+s[0:-1]+"    "+(16-len(hex2ascii(s)))*3*" "+hex2ascii(s)+"\n")
		else:
			f2.write("                "+s[0:-1]+"    "+(16-len(hex2ascii(s)))*3*" "+hex2ascii(s)+"\n")
	if "ms]" in s:
		if "<<<" in s:
			lastdir="smartcard -> interface"

		if ">>>" in s:
			lastdir="interface -> smartcard"
	s=f.readline()
f.close()
f2.close()
```

5. EL API Y EL CÓDIGO FUENTE DISTRIBUÍDO EN EL CD

En el cd vienen varias dlls e includes y varios proyectos de visual studio. De todo ello nos interesa lo que tenemos en:

<cdrom>:\HARDWARE\GEMALTO_SCR\DOCS\CTAPI.zip

Aquí tenemos documentación, librerías, includes y dos aplicaciones (en c++ y en VB) para manejar el api exportado por la dll c:\WINDOWS\system32\CTGmplus.dll previamente instalada.

Con dicho API vamos a poder conectar y enviar tramas a la tarjeta y capturar las respuestas que nos da.

Con este API nos abstraemos de la capa física y nos centramos en el protocolo de comunicación con la tarjeta.

Toda la documentación sobre comunicaciones con smartcards está en el standard ISO 7816 (http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816-4.aspx).

Junto con el api viene una aplicación de ejemplo en C++ y otra en VB. Usaremos el proyecto en C++ como base para nuestras pruebas (adjunto fuentes de las pruebas a este documento).

6. ALGUNOS CONCEPTOS SOBRE LAS SMARTCARDS.

Los protocolos de comunicación a las smartcards, la seguridad, la estructura interna, etc... está definido en el standard ISO 7816.

Para enredar con el DNIE recomiendo leer lo que se pueda del standard **ISO 7816** (hay partes que hay que soltar pasta para conseguirlas) y el standard **CWA-14890** (sobre seguridad).

El protocolo de comunicación con las tarjetas no es muy complicado, funciona mediante el intercambio de tramas (APDUs) comando – respuesta.

Las APDUs **comando** tienen el formato **CLA | INS | P1 | P2 | Lc | Data | Le** (ya depende del comando que sea que haya datos, el formato de estos, etc..).

Las APDUs **respuesta** tienen el formato **SW1 | SW2 | Data** (según la respuesta habrá datos o no).

La tarjeta implementa un sistema de archivos en el que hay DFs (dedicated files, equivalente a directorios) y EFs (elementary files, equivalente a ficheros). El directorio raiz es el MF (master file). Los archivos (tanto DFs como EFs) pueden ser identificados por nombre o por id (también por un path, que sólo es la concatenación de los ids de los DFs padre de la ruta hasta el DF o EF en cuestión). Puede ser que un archivo no tenga nombre, pero siempre tiene un id que lo identifica. **Los ids van de 0 a 65535.**

El acceso a los ficheros puede ser restringido en función de varias políticas, entre ellas que el usuario esté o no identificado mediante su pin.

Existe la posiblidad de crear canales seguros (Secure Messaging) para la comunicación con la tarjeta (algunos comandos sólo son aceptados por ésta cuando van sobre un canal seguro) mediante la encriptación de los datos intercambiados con TDES y el intercambio de la clave para TDES con Diffie-Helman (Hay otras posibilidades, pero el DNIE lo hace así).

7. LAS ZONAS DE SEGURIDAD DEL DNI ELECTRÓNICO.

Según la web http://www.dnielectronico.es/ **la información en el dni electrónico está distribuida en tres zonas:**

"ZONA PÚBLICA: Accesible en lectura sin restricciones, contenido:

Certificado CA intermedia emisora.
Claves Diffie-Hellman.
Certificado x509 de componente.

ZONA PRIVADA: Accesible en lectura por el ciudadano, mediante la utilización de la Clave Personal de Acceso o PIN, conteniendo:

Certificado de Firma (No Repudio).
Certificado de Autenticación (Digital Signature).

ZONA DE SEGURIDAD: Accesible en lectura por el ciudadano, en los Puntos de Actualización del DNIe.

Datos de filiación del ciudadano (los mismos que están en el soporte físico).
Imagen de la fotografía.
Imagen de la firma manuscrita."

7.1. LA ZONA PÚBLICA.

Como decíamos antes la tarjeta inteligente implementa un sistema de archivos en el que tenemos directorios DF y ficheros EF, y cada uno de ellos tiene asignado un identificador. **Algunos de estos DFs y EFs pueden ser seleccionados y leídos sin identificarse en la tarjeta. Esa es la zona pública.**

7.1.1. RECORRIDO DE LA ZONA PÚBLICA.

Partiendo del software de ejemplo distribuido en el cd del lector vamos a implementar una función que recorra todos los ids de 0 a 65535 sin habernos identificado mediante el pin, guardándonos los ids que han sido seleccionables (es decir, no nos ha devuelto la respuesta "no encontrado"), los datos leídos en caso de haber podido leerlos y el FCI (File control information. Cuando seleccionamos un fichero podemos enviar a la tarjeta un comando GET_RESPONSE y ésta nos devolverá alguna información adicional sobre el fichero).

(En el proyecto de C++ adjunto buscar la función OnBnClickedButtonDoTest, donde está implementada esta prueba).

Para seleccionar un fichero por id usamos la instrucción A4. La respuesta 6A 82 significa que no se ha encontrado el fichero con ese id.

Pseudocódigo:

```
i=0
ids_accesibles = []
For 0 < i < 65535:
        Comando = 00 A4 00 00 02 XX XX          ( XX XX = WORD = i )
        Respuesta = EnviarComando ( Comando )
        Si respuesta != 6A 82:
                Append ( ids_accesibles, i )
End
```

Los ids encontrados son:

815, 1280, 1536, 4415, 4608, 5439, 5456, 8032, 8288, 8448, 12640, 24928, 33120, 41312, 43104

Sabiendo que ids son seleccionables vamos a pedir el GET_RESPONSE de cada uno y a intentar hacer un READ_BINARY de los que sean EFs y se puedan acceder.

Cuando se selecciona un fichero la respuesta es del tipo:

61 XX

XX es el número de bytes que nos devolverá GET_RESPONSE.

El comando GET_RESPONSE es 00 C0 00 00 XX.

El comando READ_BINARY es 00 B0 SS SS NN:

NN número de bytes a leer.
SS SS offset del fichero en el que leer.

(Ver la función OnBnClickedButtonGetNoauthInfo en el proyecto adjunto para ver el código que realiza esta prueba).

De ellos los siguientes ids (que sí que hemos podido seleccionar) son EFs que no han podido ser leídos por políticas de seguridad:

File id = 1280

GET_RESPONSE:
0x6f 0x0c 0x85 0x0a 0x01 0x00 0x05 0x00 0x32 0x80 0xff 0xff 0xff 0xff 0x90 0x00 ?o???????2??????

File id = 4608

GET_RESPONSE:
0x6f 0x0d 0x85 0x0b 0x15 0x00 0x12 0x00 0x21 0xff 0xff 0xff 0xff 0xff 0x02 0x90 ?o???????!??????
0x00 ?

File id = 8448

GET_RESPONSE:
0x6f 0x0c 0x85 0x0a 0x25 0x00 0x21 0x02 0x58 0xb1 0xff 0x80 0xd1 0xff 0x90 0x00 ?o???%?!?X??????

File id = 41312

GET_RESPONSE:
0x6f 0x0c 0x85 0x0a 0x01 0x60 0xa1 0x00 0x20 0x80 0x80 0xff 0xff 0xff 0x90 0x00 ?o????`?? ??????

Los siguientes son DFs (se muestran con la respuesta al comando GET_RESPONSE para el mismo DF y para el padre):

File id = 4415

PARENT GET_RESPONSE:
0x6f 0x19 0x84 0x0b 0x4d 0x61 0x73 0x74 0x65 0x72 0x2e 0x46 0x69 0x6c 0x65 0x85 ?o???Master.File
0x0a 0x38 0x3f 0x00 0x00 0x0b 0xff 0xff 0xff 0xff 0xff 0x90 0x00 le??8????????

GET_RESPONSE:
0x6f 0x18 0x84 0x0a 0x49 0x43 0x43 0x2e 0x43 0x72 0x79 0x70 0x74 0x6f 0x85 0x0a ?o???ICC.Crypto?
0x38 0x3f 0x11 0x00 0x0a 0xff 0xff 0xff 0xff 0xff 0x90 0x00 to??8???????

File id = 5439

PARENT GET_RESPONSE:
0x6f 0x19 0x84 0x0b 0x4d 0x61 0x73 0x74 0x65 0x72 0x2e 0x46 0x69 0x6c 0x65 0x85 ?o???Master.File
0x0a 0x38 0x3f 0x00 0x00 0x0b 0xff 0xff 0xff 0xff 0xff 0x90 0x00 le??8????????

GET_RESPONSE:
0x6f 0x14 0x84 0x06 0x49 0x43 0x43 0x2e 0x49 0x44 0x85 0x0a 0x38 0x3f 0x15 0x00 ?o???ICC.ID??8??
0x06 0xff 0xff 0xff 0xff 0xff 0x90 0x00 ID??8???

File id = 5456

PARENT GET_RESPONSE:
0x6f 0x19 0x84 0x0b 0x4d 0x61 0x73 0x74 0x65 0x72 0x2e 0x46 0x69 0x6c 0x65 0x85 ?o???Master.File
0x0a 0x38 0x3f 0x00 0x00 0x0b 0xff 0xff 0xff 0xff 0xff 0x90 0x00 le??8????????

GET_RESPONSE:
0x6f 0x1a 0x84 0x0c 0xa0 0x00 0x00 0x00 0x63 0x50 0x4b 0x43 0x53 0x2d 0x31 0x35 ?o???????cPKCS-1
0x85 0x0a 0x38 0x50 0x15 0x00 0x0c 0xff 0xff 0xff 0xff 0xff 0x90 0x00 15??8P????????

File id = 12640

PARENT GET_RESPONSE:
0x6f 0x19 0x84 0x0b 0x4d 0x61 0x73 0x74 0x65 0x72 0x2e 0x46 0x69 0x6c 0x65 0x85 ?o???Master.File

0x0a 0x38 0x3f 0x00 0x00 0x0b 0xff 0xff 0xff 0xff 0xff 0x90 0x00 le??8????????

GET_RESPONSE:
0x6f 0x18 0x84 0x0a 0x44 0x4e 0x49 0x65 0x2e 0x41 0x64 0x6d 0x69 0x6e 0x85 0x0a ?o???DNIe.Admin?
0x38 0x60 0x31 0x00 0x0a 0xff 0xff 0xff 0xff 0xff 0x90 0x00 in??8`1?????

File id = 24928

PARENT GET_RESPONSE:
0x6f 0x19 0x84 0x0b 0x4d 0x61 0x73 0x74 0x65 0x72 0x2e 0x46 0x69 0x6c 0x65 0x85 ?o???Master.File
0x0a 0x38 0x3f 0x00 0x00 0x0b 0xff 0xff 0xff 0xff 0xff 0x90 0x00 le??8????????

GET_RESPONSE:
0x6f 0x16 0x84 0x08 0x44 0x4e 0x49 0x65 0x2e 0x50 0x75 0x62 0x85 0x0a 0x38 0x60 ?o???DNIe.Pub??8
0x61 0x00 0x08 0xe0 0xff 0xff 0xff 0xff 0x90 0x00 ub??8`a???

File id = 33120

PARENT GET_RESPONSE:
0x6f 0x19 0x84 0x0b 0x4d 0x61 0x73 0x74 0x65 0x72 0x2e 0x46 0x69 0x6c 0x65 0x85 ?o???Master.File
0x0a 0x38 0x3f 0x00 0x00 0x0b 0xff 0xff 0xff 0xff 0xff 0x90 0x00 le??8????????

GET_RESPONSE:
0x6f 0x17 0x84 0x09 0x44 0x4e 0x49 0x65 0x2e 0x50 0x72 0x69 0x76 0x85 0x0a 0x38 ?o???DNIe.Priv??
0x60 0x81 0x00 0x09 0xe0 0xff 0xff 0xff 0xff 0x90 0x00 iv??8`?????

Todos estos DFs tienen estos nombres:

4415, ICC.Crypto
5439, ICC.ID
5456, cPKCS-115
12640, DNIe.Admin
24928, DNIe.Pub
33120, DNIe.Priv

Todos cuelgan del Master.File, que siempre tiene id 0x3f:

File name = Master.File

GET_RESPONSE:
0x6f 0x19 0x84 0x0b 0x4d 0x61 0x73 0x74 0x65 0x72 0x2e 0x46 0x69 0x6c 0x65 0x85 ?o???Master.File
0x0a 0x38 0x3f 0x00 0x00 0x0b 0xff 0xff 0xff 0xff 0xff 0x90 0x00 le??8????????

A continuación están los EFs que sí hemos podido leer (se muestran con la respuesta al comando GET_RESPONSE y READ_BINARY):

File id = 815

GET_RESPONSE:
0x6f 0x0c 0x85 0x0a 0x01 0x2f 0x03 0x00 0x28 0x00 0x80 0xff 0xff 0xff 0x90 0x00 ?o????/??(??????

READ_BINARY:
0x44 0x4e 0x49 0x65 0x20 0x30 0x31 0x2e 0x31 0x33 0x20 0x41 0x31 0x31 0x20 0x48 ?DNIe 01.13 A11
0x20 0x34 0x43 0x33 0x34 0x20 0x45 0x58 0x50 0x20 0x31 0x2d 0x28 0x28 0x34 0x2e H 4C34 EXP 1-((4
0x32 0x2d 0x35 0x29 0x29 0x00 0x00 0x00 0x90 0x00 1-((4.2-5)

File id = 1536

GET_RESPONSE:
0x6f 0x0c 0x85 0x0a 0x01 0x00 0x06 0x00 0x19 0x00 0xff 0xff 0xff 0xff 0x90 0x00 ?o??????????????

READ_BINARY:

El IDESP. Varía entre cada DNI.

File id = 8032

GET_RESPONSE:
0x6f 0x0c 0x85 0x0a 0x01 0x60 0x1f 0x03 0x25 0x00 0xff 0xff 0xff 0xff 0x90 0x00 ?o????`??%??????

READ_BINARY:

Certificado. Varía entre cada DNI.

File id = 8288

GET_RESPONSE:
0x6f 0x0c 0x85 0x0a 0x01 0x60 0x20 0x04 0x2c 0x00 0xff 0xff 0xff 0xff 0x90 0x00 ?o????` ?,??????

READ_BINARY:
0x30 0x82 0x04 0x28 0x30 0x82 0x03 0x91 0xa0 0x03 0x02 0x01 0x02 0x02 0x10 0x1a ?0??(0??????????
0xed 0x35 0x7a 0x7b 0xa8 0xc8 0x7b 0x43 0xfa 0xdf 0x3f 0xa9 0x77 0x56 0x80 0x30 ??5z{??{C????wV?
0x0d 0x06 0x09 0x2a 0x86 0x48 0x86 0xf7 0x0d 0x01 0x01 0x05 0x05 0x00 0x30 0x81 0???*?H????????0
0x87 0x31 0x0b 0x30 0x09 0x06 0x03 0x55 0x04 0x06 0x13 0x02 0x45 0x53 0x31 0x28 ??1?0???U????ES1
0x30 0x26 0x06 0x03 0x55 0x04 0x0a 0x0c 0x1f 0x44 0x49 0x52 0x45 0x43 0x43 0x49 (0&??U????DIRECC
0x4f 0x4e 0x20 0x47 0x45 0x4e 0x45 0x52 0x41 0x4c 0x20 0x44 0x45 0x20 0x4c 0x41 ION GENERAL DE L
0x20 0x50 0x4f 0x4c 0x49 0x43 0x49 0x41 0x31 0x0d 0x30 0x0b 0x06 0x03 0x55 0x04 A POLICIA1?0???U
0x0b 0x0c 0x04 0x44 0x4e 0x49 0x45 0x31 0x1c 0x30 0x1a 0x06 0x03 0x55 0x04 0x0b ????DNIE1?0???U?
0x0c 0x13 0x41 0x43 0x20 0x52 0x41 0x49 0x5a 0x20 0x43 0x4f 0x4d 0x50 0x4f 0x4e ???AC RAIZ COMPO
0x45 0x4e 0x54 0x45 0x53 0x31 0x21 0x30 0x1f 0x06 0x03 0x55 0x04 0x03 0x0c 0x18 NENTES1!0???U???
0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x36 0x35 0x37 0x33 0x35 0x32 0x34 0x34 ?000000006573524
0x34 0x39 0x36 0x30 0x30 0x30 0x30 0x36 0x30 0x1e 0x17 0x0d 0x30 0x36 0x30 0x32 4496000060???060
0x32 0x31 0x30 0x39 0x33 0x37 0x30 0x33 0x5a 0x17 0x0d 0x33 0x31 0x30 0x32 0x32 221093703Z??3102
0x30 0x31 0x37 0x31 0x33 0x31 0x35 0x5a 0x30 0x72 0x31 0x0b 0x30 0x09 0x06 0x03 20171315Z0r1?0??
0x55 0x04 0x06 0x13 0x02 0x45 0x53 0x31 0x28 0x30 0x26 0x06 0x03 0x55 0x04 0x0a ?U????ES1(0&??U?
0x0c 0x1f 0x44 0x49 0x52 0x45 0x43 0x43 0x49 0x4f 0x4e 0x20 0x47 0x45 0x4e 0x90 ???DIRECCION GEN
0x00 ?
0x45 0x52 0x41 0x4c 0x20 0x44 0x45 0x20 0x4c 0x41 0x20 0x50 0x4f 0x4c 0x49 0x43 ?ERAL DE LA POLI
0x49 0x41 0x31 0x0d 0x30 0x0b 0x06 0x03 0x55 0x04 0x0b 0x0c 0x04 0x44 0x4e 0x49 CIA1?0???U????DN
0x45 0x31 0x0d 0x30 0x0b 0x06 0x03 0x55 0x04 0x0b 0x0c 0x04 0x46 0x4e 0x4d 0x54 IE1?0???U????FNM
0x31 0x1b 0x30 0x19 0x06 0x03 0x55 0x04 0x03 0x0c 0x12 0x41 0x43 0x20 0x43 0x4f T1?0???U????AC C
0x4d 0x50 0x4f 0x4e 0x45 0x4e 0x54 0x45 0x53 0x20 0x30 0x30 0x31 0x30 0x81 0x9f OMPONENTES 0010?
0x30 0x0d 0x06 0x09 0x2a 0x86 0x48 0x86 0xf7 0x0d 0x01 0x01 0x01 0x05 0x00 0x03 ?0???*?H????????
0x81 0x8d 0x00 0x30 0x81 0x89 0x02 0x81 0x81 0x00 0xde 0x3c 0xc3 0x0b 0x66 0x8c ????0???????<??f
0xb2 0x35 0x34 0x85 0xfd 0x42 0xa1 0x19 0x21 0x25 0xf4 0xa8 0x25 0x04 0x19 0x7e ??54??B??!%??%??
0xbd 0x3c 0xb6 0xab 0xff 0xb8 0xe6 0x8f 0x2e 0x84 0x29 0x3b 0xb7 0x25 0xfd 0x61 ??<??????.?);?%?
0x8f 0xd1 0x0f 0xda 0x1a 0x40 0x97 0x37 0xd7 0x4b 0xd1 0xb9 0x79 0x84 0x68 0x51 a?????@??7?K??y?h
0x56 0xce 0xff 0x55 0xf5 0xe3 0x52 0x5b 0xc5 0x8b 0xfd 0x37 0x99 0xc2 0xa7 0xe3 QV??U??R[???????
0x00 0x47 0x1d 0x93 0x03 0x10 0x5e 0xe8 0xec 0x4c 0x67 0xd4 0x76 0xdc 0x48 0xd1 ??G????^??Lg?v?H
0x83 0x46 0x70 0xc2 0x7e 0xbf 0x46 0x68 0x77 0x9d 0x7b 0x85 0x5b 0x44 0xaf 0xa7 ??Fp???Fhw?{?[D?
0x33 0xda 0x45 0x1b 0xb9 0x54 0x99 0x4a 0x71 0x89 0x10 0xfd 0xe6 0x31 0x3a 0xab ?3?E??T?Jq????1:
0x62 0x66 0x31 0x3f 0xdb 0xd1 0x56 0x4f 0xa5 0x7d 0x02 0x03 0x01 0x00 0x01 0xa3 ?bf1???VO?}?????
0x82 0x01 0xa7 0x30 0x82 0x01 0xa3 0x30 0x12 0x06 0x03 0x55 0x1d 0x13 0x01 0x90 ????0???0???U???
0x00 ?
0x01 0xff 0x04 0x08 0x30 0x06 0x01 0x01 0xff 0x02 0x01 0x00 0x30 0x1d 0x06 0x03 ?????0???????0??
0x55 0x1d 0x0e 0x04 0x16 0x04 0x14 0x3f 0x32 0x0e 0x97 0x94 0xb8 0xf9 0x56 0x6e ?U???????2?????V
0xa3 0xd7 0x21 0xa6 0x17 0x54 0xfa 0x92 0x3a 0x12 0xb1 0x30 0x1f 0x06 0x03 0x55 n??!??T??:??0???
0x1d 0x23 0x04 0x18 0x30 0x16 0x80 0x14 0x45 0xd7 0x64 0x65 0xf2 0x25 0x04 0xd8 U?#??0???E?de?%?
0x04 0x5e 0x66 0x27 0x41 0x9d 0x76 0x50 0x05 0xa2 0xd1 0x58 0x30 0x0e 0x06 0x03 ??^f'A?vP???X0??
0x55 0x1d 0x0f 0x01 0x01 0xff 0x04 0x04 0x03 0x02 0x01 0x06 0x30 0x37 0x06 0x03 ?U???????????07?
0x55 0x1d 0x20 0x04 0x30 0x30 0x2e 0x30 0x2c 0x06 0x04 0x55 0x1d 0x20 0x00 0x30 ?U? ?00.0,??U? ?
0x24 0x30 0x22 0x06 0x08 0x2b 0x06 0x01 0x05 0x05 0x07 0x02 0x01 0x16 0x16 0x68 0$0"??+?????????
0x74 0x74 0x70 0x3a 0x2f 0x2f 0x77 0x77 0x77 0x2e 0x64 0x6e 0x69 0x65 0x2e 0x65 http://www.dnie.
0x73 0x2f 0x64 0x70 0x63 0x30 0x82 0x01 0x02 0x06 0x03 0x55 0x1d 0x1f 0x04 0x81 es/dpc0?????U???
0xfa 0x30 0x81 0xf7 0x30 0x81 0xf4 0xa0 0x81 0xf1 0xa0 0x81 0xee 0x86 0x23 0x68 ??0??0?????????#
0x74 0x74 0x70 0x3a 0x2f 0x2f 0x63 0x72 0x6c 0x73 0x2e 0x64 0x6e 0x69 0x65 0x2e http://crls.dnie
0x65 0x73 0x2f 0x63 0x72 0x6c 0x73 0x2f 0x41 0x52 0x4c 0x43 0x4f 0x4d 0x2e 0x63 .es/crls/ARLCOM.
0x72 0x6c 0x86 0x81 0xc6 0x6c 0x64 0x61 0x70 0x3a 0x2f 0x2f 0x6c 0x64 0x61 0x70 crl???ldap://lda
0x2e 0x64 0x6e 0x69 0x65 0x2e 0x65 0x73 0x2f 0x43 0x4e 0x3d 0x43 0x52 0x4c 0x2c p.dnie.es/CN=CRL
0x43 0x4e 0x3d 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x36 0x35 0x37 0x33 0x90 ,CN=000000006573
0x00 N
0x35 0x32 0x34 0x34 0x34 0x39 0x36 0x30 0x30 0x30 0x30 0x36 0x2c 0x4f 0x55 0x3d ?524449600006,OU
0x41 0x43 0x25 0x32 0x30 0x52 0x41 0x49 0x5a 0x25 0x32 0x30 0x43 0x4f 0x4d 0x50 =AC%20RAIZ%20COM
0x4f 0x4e 0x45 0x4e 0x54 0x45 0x53 0x2c 0x4f 0x55 0x3d 0x44 0x4e 0x49 0x45 0x2c PONENTES,OU=DNIE
0x4f 0x3d 0x44 0x49 0x52 0x45 0x43 0x43 0x49 0x4f 0x4e 0x25 0x32 0x30 0x47 0x45 ,O=DIRECCION%20G
0x4e 0x45 0x52 0x41 0x4c 0x25 0x32 0x30 0x44 0x45 0x25 0x32 0x30 0x4c 0x41 0x25 ENERAL%20DE%20LA
0x32 0x30 0x50 0x4f 0x4c 0x49 0x43 0x49 0x41 0x2c 0x43 0x3d 0x45 0x53 0x3f 0x61 %20POLICIA,C=ES?a
0x75 0x74 0x68 0x6f 0x72 0x69 0x74 0x79 0x52 0x65 0x76 0x6f 0x63 0x61 0x74 0x69 authorityRevocat

```
0x6f 0x6e 0x4c 0x69 0x73 0x74 0x3f 0x62 0x61 0x73 0x65 0x3f 0x6f 0x62 0x6a 0x65     ionList?base?obj
0x63 0x74 0x63 0x6c 0x61 0x73 0x73 0x3d 0x63 0x52 0x4c 0x44 0x69 0x73 0x74 0x72     ectclass=cRLDist
0x69 0x62 0x75 0x74 0x69 0x6f 0x6e 0x50 0x6f 0x69 0x6e 0x74 0x30 0x0d 0x06 0x09     ributionPoint0??
0x2a 0x86 0x48 0x86 0xf7 0x0d 0x01 0x01 0x05 0x05 0x00 0x03 0x81 0x81 0x00 0x8f     ?*?H????????????
0x83 0x7a 0xd2 0xf2 0xa8 0xc4 0xba 0x6a 0xb4 0x5e 0x0d 0x4b 0x39 0x2d 0x6e 0xc0     ??z?????j?^?K9-n
0xe6 0x6b 0x9d 0xa3 0xf9 0x7e 0xfb 0x08 0xda 0x7d 0xc3 0xbb 0xd3 0xa6 0x54 0x9b     ??k???????}????T
0xe1 0x04 0x71 0xff 0x7f 0x28 0x20 0x9a 0x17 0xf0 0x71 0x7a 0xf6 0xf6 0xd5 0x1e     ???q??( ???qz???
0x9f 0x30 0xdf 0xd5 0x8f 0xe8 0x04 0xf0 0x2b 0xf2 0xb2 0x58 0x20 0xde 0xf5 0x95     ??0??????+??X ??
0xc2 0xed 0x85 0xe3 0xb0 0x80 0x38 0x04 0x38 0xd8 0x24 0x70 0x21 0x80 0x46 0x90     ???????8?8?$p!?F
0x00                                                                                ?
0x26 0x67 0xf2 0x9d 0x75 0xb3 0x0c 0xb9 0xf2 0xbb 0x4a 0x03 0xdd 0xa5 0x8d 0xe2     ?&g??u?????J????
0xc1 0x50 0xed 0x31 0x6a 0xb9 0xb2 0x9c 0xa6 0x32 0x10 0x01 0x7d 0xd7 0x0f 0xfc     ??P?1j????2??}??
0x21 0x31 0x6f 0x88 0xf7 0x53 0x90 0x4d 0x74 0x3b 0x70 0x21 0x45 0x23 0x76 0x0a     ?!1o??S?Mt;p!E#v
0x90 0x00                                                                           o?
```

File id = 43104

GET_RESPONSE:

```
0x6f 0x0c 0x85 0x0a 0x01 0x60 0xa8 0x00 0xb5 0x00 0xff 0xff 0xff 0xff 0x90 0x00     ?o????`?????????
```

READ_BINARY:

Otros datos (sin interpretar). Varían entre cada DNI.

El fichero con ID 815 contiene la **versión del DNIE**. Estos datos varían con cada DNIE si cambia la versión de éste.

En la lectura de arriba: DNIe 01.13 A11H 4C34 EXP 1-((4.2-5)

En otro DNIE: DNIe 01.13 B11H 4C34 EXP 1-(3.5-3)

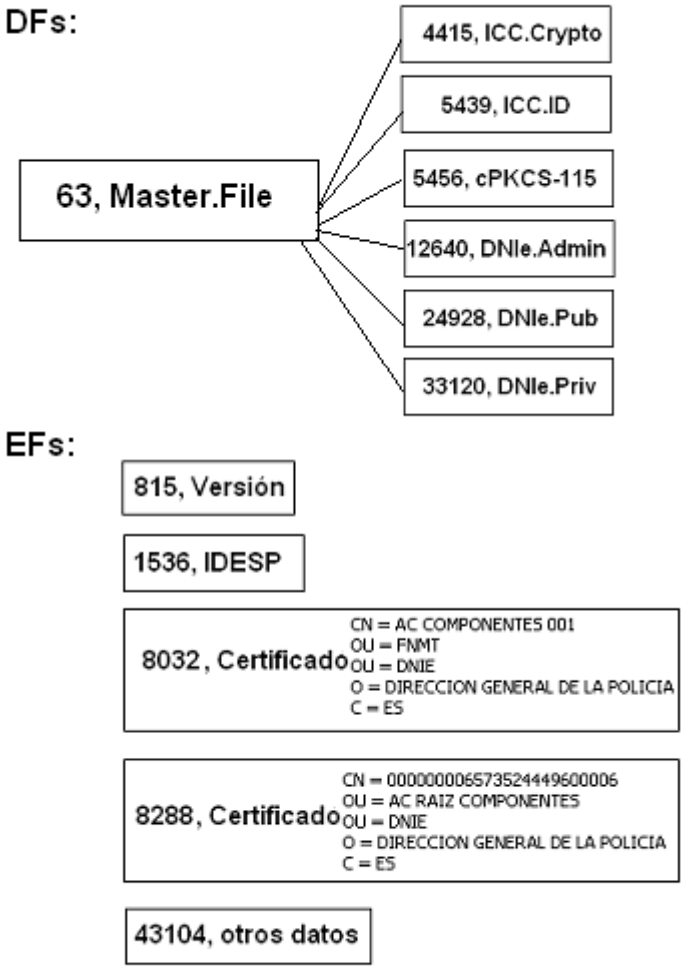
El fichero con ID 1536 contiene el **IDESP** del DNIE. Este dato se puede ver en la carátula del DNIE también.

El fichero con ID 8032 contiene un certificado (un **certificado de componente**) que es leído y usado por la interfaz cuando se va a establecer el canal seguro con la tarjeta. Este certificado varía entre cada DNIE.

El fichero con ID 8288 también contiene un certificado (un **certificado de la CA de componente**), que no varía entre DNIE.

El fichero con ID 43104 contiene 0xB0 bytes de datos (cuyo significado desconozco) que también varían entre cada DNIE.

Datos públicos accesibles:



7.2. LA ZONA PRIVADA.

Para acceder a la zona privada es necesario introducir el pin de usuario a la tarjeta con el comando VERIFY.

El comando VERIFY tiene un formato sencillo: 00 20 00 00 Lc Data

En data iría el PIN.

Sin embargo el DNIE restringe el uso de algunos comandos si estos no se envían sobre un canal seguro (usando Secure Messaging).

7.2.1. EL ESTABLECIMIENTO DEL CANAL SEGURO.

Sobre el uso de Secure Messaging y el establecimiento del canal seguro, leer los standares **ISO 7816–8 y CWA 14890**. La normativa ISO 7816–8 es de pago y al menos yo no he encontrado nada público en internet así que basicamente he tirado del CWA 14890, disponible en el cd o en la web del dni electrónico.

Lo primero, hacemos una captura con el sniffer del establecimiento de canal seguro realizado por el software distribuido en el cd (el plugin para los navegadores) para luego ir analizándola en base a la documentación.

Trás realizar la captura implementamos una función en la que intentamos reproducir los pasos realizados por el software distribuido para probarlo con distintos DNIEs y comparar, ver hasta donde se puede llegar, donde cambia la captura cuando se hace con dos DNIEs diferentes, etc... Con esta aplicación veremos que se puede reproducir los pasos hasta un punto donde se pide un challenge a la smartcard cuyo valor (aleatorio) no podemos controlar.

Lo siguiente que sería interesante hacer es implementar el establecimiento de canal seguro en nuestra aplicación, para poder leer el contenido de los demás ficheros (habiendo introducido también el pin), ya que a partir de que se establece el canal seguro no podemos ver nada en la captura, todo va cifrado con TDES. Aún no he sido capaz de implementar dicho establecimiento.

7.2.1.1. EL ANÁLISIS DE LA CAPTURA DEL INTERCAMBIO DE APDUS DEL SOFTWARE DISTRIBUIDO CON LA TARJETA EN EL ESTABLECIMIENTO DEL CANAL SEGURO

A continuación va un análisis de los datos capturados con usbsnoop en el establecimiento de un canal seguro para secure messaging en base a la documentación CWA 14890 (página 41).

A) SELECT_FILE (id = 8032, 0x1F60)

APDU: [CLA = 00] [INS = A4] [P1=00] [P2=00] [Lc=02] [Id = 60 1F]

B) READ_BINARY

APDU: [CLA = 00] [INS = B0] [Offset=00 00] [Length=FF]
APDU: [CLA = 00] [INS = B0] [Offset=00 FF] [Length=FF]
APDU: [CLA = 00] [INS = B0] [Offset=02 FD] [Length=28]

Ha leído todo el contenido del fichero 8032 con tres comandos READ_BINARY. Si volcamos todo ese contenido a un fichero y lo renombramos a .cer podemos ver que es un certificado X.509 (certificado de componente). Este certificado es diferente para cada DNI.

X.509:
- Versión:
  - V3
- Algoritmo de firma:
  - Sha1RSA
- Emisor:
  - CN = AC COMPONENTES 001
  - OU = FNMT
  - OU = DNIE
  - O = DIRECCION GENERAL DE LA POLICIA
  - C = ES
- Válido desde:
  - miércoles, 21 de octubre de 2009 14:20:31
- Válido hasta:
  - miércoles, 21 de octubre de 2020 14:20:31
- Acceso a la información de entidad emisora:
  - [1]Acceso a información de autoridad
    - Método de acceso=Emisor de la entidad emisora de certificados (1.3.6.1.5.5.7.48.2)
    - Nombre alternativo:
      - Dirección URL=http://www.dnie.es/certs/ACRaiz.crt
- Puntos de distribución CLR:

[1]Punto de distribución CRL
Nombre del punto de distribución:
Nombre completo:
Dirección URL=http://crls.fnmt.dnie.es/crls/CRLC15D932112B29342770C8AA34A5A237326CC982C.crl
Restricciones básicas:
Tipo de asunto=Entidad final
Restricción de longitud de ruta=Ninguno
Uso de la clave:
Firma digital, Contrato de claves (88)
Algoritmo de identificación:
Sha1

C) MANAGE_SECURITY_ENVIRONMENT – Set for verification

APDU: [CLA = 00]
[INS = 22]
[P1=81] (Set for verification)
[Control Reference Template, CRT]

CRT: [B6] (Seleccionar clave privada para las próximas verificaciones).
[Lb6 = 04]
[83]
[L83 = 02]
[Keyref = 02 0f]

Con este comando está diciendo a la tarjeta que deje seleccionada una clave conocida por ella para las próximas operaciones.

Según la documentación (CWA 14890) aquí se está seleccionando la PK.RCA.AUT (public key, root CA, authentication).

D) PERFORM_SECURITY_OPERATION – verify certificate

APDU: [CLA = 00]
[INS = 2A]
[P1 = 00]
[P2 = AE] (opción VERIFY_CERTIFICATE)
[Lc = D2]
[Card Verifiable Certificate, CVC]

Con este comando estamos pidiendo a la tarjeta que verifique un certificado en formato CVC.

Los certificados CVC no contienen algunas partes que sí contienen los X.509. Es un formato más manejable por la tarjeta, específico para la comprobación en ésta. Por ejemplo no tienen las partes en texto plano o las fechas de validez (dato difícil de comprobar por la tarjeta).

El CVC enviado:

[7f 21 81 ce] (identificador de CVC, longitud 0xCE)
[5f 37 81 80] (Signature del certificado, longitud 0x80, 1024 bits)
[3c ba dc 36 84 be f3 20 41 ad 15 50 89 25 8d fd
20 c6 91 15 d7 2f 9c 38 aa 99 ad 6c 1a ed fa b2
bf ac 90 92 fc 70 cc c0 0c af 48 2a 4b e3 1a fd
bd 3c bc 8c 83 82 cf 06 bc 07 19 ba ab b5 6b 6e
c8 07 60 a4 a9 3f a2 d7 c3 47 f3 44 27 f9 ff 5c
8d e6 d6 5d ac 95 f2 f1 9d ac 00 53 df 11 a5 07
fb 62 5e eb 8d a4 c0 29 9e 4a 21 12 ab 70 47 58
8b 8d 6d a7 59 22 14 f2 db a1 40 c7 d1 22 57 9b] (Signature)
[5f 38 3d]

[22 53 c8 b9 cb 5b c3 54 3a 55 66 0b da 80 94 6a
fb 05 25 e8 e5 58 6b 4e 63 e8 92 41 49 78 36 d8
d3 ab 08 8c d4 4c 21 4d 6a c8 56 e2 a0 07 f4 4f
83 74 33 37 37 1a dd 8e 03](PK_part2¿?)
[00 01 00 01](PK_exp¿?)
[42 08] (CAR)
[65 73 52 44 49 60 00 06] (esRDI)

Este CVC es igual para dos DNIEs diferentes, viene de fuera.

Una vez descifrados estos datos, habrá otra serie de datos útiles debajo de la encriptación para continuar con el proceso de establecimiento del canal seguro.

En estos datos cifrados: 0x6A || CPI || CAR || CHR || CHA || OID || PK_part1 || Hash || 0xBC

Según la documentación el certificado que se está verificando es C_CV.CA.CS_AUT (card verifiable certificate, CA, cert signing authentication). Se va a verificar con la clave puesta con el comando anterior, PK.RCA.AUT. Si se verifica con éxito la tarjeta almacena la PK contenida en el CVC verificado y sirve para importar otros certificados.

E) MANAGE_SECURITY_ENVIRONMENT – Set for verification

APDU: [CLA = 00]
[INS = 22]
[P1=81] (Set for verification)
[Control Reference Template, CRT]

CRT: [B6] (Seleccionar clave privada para las próximas verificaciones).
[Lb6 = 0A]
[83]
[L83 = 08]
[Keyref = 65 73 53 44 49 60 00 06 = esSDI...]

Con este comando está diciendo a la tarjeta que deje seleccionada una clave conocida por ella (el keyref hace referencia a una clave esSDI) para las próximas operaciones. Creo que la clave seleccionada es la que va en el CVC verificado en el comando anterior. Según la documentación (CWA 14890) aquí se está seleccionando la PK.CAifd.AUT (public key, CA de la interfaz, authentication).

F) PERFORM_SECURITY_OPERATION – verify certificate

APDU: [CLA = 00]
[INS = 2A]
[P1 = 00]
[P2 = AE] (opción VERIFY_CERTIFICATE)
[Lc = D1]
[Card Verifiable Certificate, CVC]

Con este comando estamos pidiendo a la tarjeta que verifique un certificado en formato CVC.

El CVC enviado:

[7f 21 81 cd] (identificador de CVC, longitud 0xCE)
[5f 37 81 80] (Signature del certificado, longitud 0x80, 1024 bits)
[82 5b 69 c6 45 1e 5f 51 70 74 38 5f 2f 17 d6 4d
fe 2e 68 56 75 67 09 4b 57 f3 c5 78 e8 30 e4 25
57 2d e8 28 fa f4 de 1b 01 c3 94 e3 45 c2 fb 06

29 a3 93 49 2f 94 f5 70 b0 0b 1d 67 77 29 f7 55
d1 07 02 2b b0 a1 16 e1 d7 d7 65 9d b5 c4 ac 0d
de ab 07 ff 04 5f 37 b5 da f1 73 2b 54 ea b2 38
a2 ce 17 c9 79 41 87 75 9c ea 9f 92 a1 78 05 a2
7c 10 15 ec 56 cc 7e 47 1a 48 8e 6f 1b 91 f7 aa] (Signature)
[5f 38 3c]
[ad fc 12 e8 56 b2 02 34 6a f8 22 6b 1a 88 21 37
dc 3c 5a 57 f0 d2 81 5c 1f cd 4b b4 6f a9 15 7f
df fd 79 ec 3a 10 a8 24 cc c1 eb 3c e0 b6 b4 39
6a e2 36 59 00 16 ba 69](PK_part2¿?)
00 01 00 01(PK_exp¿?)
[42 08] (CAR)
65 73 53 44 49 60 00 06 (esSDI)

Este CVC es igual para dos DNIEs diferentes, viene de fuera.

Una vez descifrados estos datos, habrá otra serie de datos útiles debajo de la encriptación para continuar con el proceso de establecimiento del canal seguro.

En estos datos cifrados: 0x6A || CPI || CAR || CHR || CHA || OID || PK_part1 || Hash || 0xBC

Según la documentación el certificado que se está verificando es C_CV.IFD.AUT (card verifiable certificate, interfaz, authentication). La tarjeta verifica este CVC con la clave PK.CAifd.CS_AUT.

El certificado verificado contiene la clave pública de la interfaz PK.IFD.AUT, la cual es almacenada temporalmente en la tarjeta.

G) MANAGE_SECURITY_ENVIRONMENT-Set for internal and external authentication

APDU: [CLA = 00]
[INS = 22]
[P1=C1] (Set for internal and external authentication)
[CRT (SK.ICC.AUT) y CRT(PK.IFD.AUT) = 12 84 02 02 1f 83 0c 00 00 00
00 20 00 00 00 00 00 00 01]

En los suguientes pasos va a realizarse un INTERNAL_AUTHENTICATE y un EXTERNAL_AUTHENTICATE, pero previamente debe seleccionarse la clave secreta interna y la clave pública externa que va a usar la tarjeta.

CRT (SK.ICC.AUT): [84] [L=02] [keyref para SK.ICC.AUT = 02 1f]
CRT(PK.IFD.AUT): [83] [L=0c] [keyref para PK.IFD.AUT = 00 00 00 00 20 00 00 00 00 00 00 01]

La PK.IFD.AUT es la contenida en el CVC enviado con el comando anterior.

J) INTERNAL_AUTHENTICATE

APDU: [CLA = 00]
[INS = 88]
[P1=00]
[P2=00]
[Lc=10]
[Data= Challenge = 7b 1d de 38 28 bc 70 ec 20 00 00 00 00 00 00 01]

Con este comando la interfaz manda un challenge a la tarjeta.

El challenge se compone de RND.IFD || SN.IFD, es decir, un número aleatorio generado por la interfaz de 8 bytes, seguido por otros 8 bytes que son el SN.IFD (serialNumber.interfaz), que como vemos es el mismo que se usó en el comando anterior para el CRT(PK.IFD.AUT). Según la documentación el SN.IFD es tomado del campo CHR del CVC.

La tarjeta computará la firma sobre el challenge con su clave secreta SK.ICC.AUT y se la devolverá al IFD encriptada con PK.IFD.AUT.

El IFD verifica la respuesta con la PK.ICC.AUT (que es la pública asociada a la SK.ICC.AUT seleccionada en el comando anterior).

Exactamente la respuesta de la ICC será lo siguiente:

Response = E(PK.IFD.AUT)(SIGMIN)

SIGMIN = min(SIG, N.ICC – SIG)

SIG = DS [SK.ICC.AUT](0x6A || PRND1 || Kicc || h(PRND1 || Kicc || C) || 0xBC)

"min" es un cálculo realizado para que la encriptación con PK.IFD.AUT sea correcta y que cuando la interfaz descrifre el mensaje con SK.IFD.AUT sólo de un resultado.

PRND1 es un valor aleatorio generado, y h(x) es un hash calculado.

De todo lo que compone el SIG lo que más nos interesa es el Kicc que va a ser parte de las claves para el cifrado TDES del canal seguro.

Nota: La respuesta logeada es de 0x80 bytes y es recuperada con el comando GET_RESPONSE, 00 0C 00 00 80.

K) GET_CHALLENGE

APDU: [CLA = 00]
[INS = 84]
[P1 = 00]
[P2 = 00]
[Data = 08] (longitud del challenge que se pide)

A este mensaje la tarjeta contesta con un challenge de 8 bytes generado aleatoriamente. En la captura analizada:

challenge = [00 ef 5e 32 c3 84 33 9d a1]

L) EXTERNAL_AUTHENTICATE

APDU: [CLA = 00]
[INS = 82]
[P1 = 00]
[P2 = 00]
[Data = E[PK.ICC.AUT](SIGMIN)]

La interfaz computa la firma sobre la concatenación del challenge que le ha enviado la tarjeta con su propia clave generada Kifd y algunos datos más (esta clave junto a Kicc será lo que se use para el cifrado TDES del canal seguro), usando la clave secreta SK.IFD.AUT (asociada a PK.IFD.AUT), y además lo encriptará con PK.ICC.AUT para enviársela a la tarjeta.

(Nota: la tarjeta verificará que Kifd es distinto de Kicc, sino dará un error).

E[PK.ICC.AUT](SIGMIN)

SIGMIN = min(SIG, N.IFD – SIG)

SIG = DS[SK.IFD.AUT](0x6A || PRND || Kifd || h(PRND || Kifd || RND.ICC || SN.ICC) || 0xBC)

(Nota: cuando se importó la PK.IFD.AUT en el paso F, que se extrajo del CVC, también se extrajo el "rol" de la interfaz y se guardó temporalmente, es decir, se extrajo del CVC qué cosas puede hacer la interfaz que se ha autenticado con esta clave pública.

A partir de aquí el canal seguro está establecido, los datos necesarios para el cifrado de este canal son Kicc y Kifd.

Para calcular las claves para TDES primero se xorean Kicc y Kifd (cada uno tiene 32 bytes).

Kifd-icc = Kifd xor Kicc

Con este valor se calculan las claves (la función hash hsm es SHA-1):

HASH1 = hsm(Kifd-icc || c=1)

HASH2 = hsm(Kifd-icc || c=2)

Los bytes 1...8 de HASH1 son Ka(ENC) y los bytes 9...16 son Kb(ENC).

Los bytes 1...8 de HASH2 son Ka(MAC) y los bytes 9...16 son Kb(MAC).

Entonces a partir de Kifd-icc obtenemos 4 claves de 8 bytes cada una necesarias para el cifrado TDES.

K = Ka(ENC) || Kb(ENC) || Ka(MAC) || Kb(MAC)

7.2.1.2. DESARROLLO DE CÓDIGO PARA ESTABLECIMIENTO DEL CANAL SEGURO

Para la implementación del establecimiento del canal seguro necesitamos conocer:

1. El par PK.IFD.AUT – SK.IFD.AUT.
2. La pública PK.ICC.AUT.
3. El algoritmo usado para calcular las signatures.
4. El algoritmo usado para los cifrados con clave pública.

Teniendo estos datos podemos enviar todos los comandos exactamente igual que en la captura hasta el punto K (en el apartado anterior), en el que recibimos en challenge de la tarjeta y debemos computar la firmar con la SK.IFD.AUT.

7.2.1.3. HOOKS EN EL SOFTWARE DISTRIBUIDO PARA CAPTURAR EL TRÁFICO CIFRADO

Implementar el establecimiento del canal seguro y cifrado para Secure Messaging puede acabar siendo una tarea bastante tediosa, además de haber poca documentación al respecto (qué pares de claves usar, que algoritmos espera el DNIE que sea usados, etc...). A esto le sumo que no soy un experto en temas de criptografía. Así que finalmente me decanto por empezar a desensamblar la dll que se carga con internet

explorer y que se encarga de logearse en la tarjeta y leer los datos privados cuando hace falta identificarse en alguna web que soporte identificación por DNIe.

Encontramos en el binario varias cadenas llamativas:

```
db '.?AV?$TwoBases@V?$SimpleKeyedTransformation@VBlockTransformation@'
db 'CryptoPP@@@CryptoPP@@UDES_EDE3_Info@2@@CryptoPP@@',0
align 4
```

El software compila con la librería cryptopp (http://www.cryptopp.com).

Vamos al fuente des.cpp de cryptopp (http://www.trolocsis.com/crypto++/des_8cpp-source.html). En seguida encontramos la parte en ensamblador equivalente al fuente:

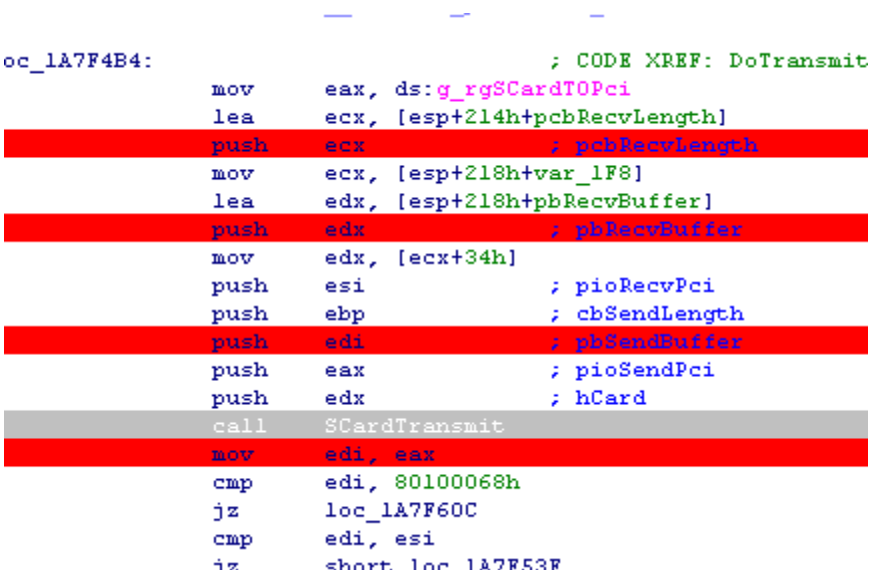
```
.text:01A5D8D0
.text:01A5D8D0 CryptoPP_RawDes_IPERM proc near
.text:01A5D8D0
.text:01A5D8D0                  mov     edx, [ecx]
.text:01A5D8D2                  rol     edx, 4
.text:01A5D8D5                  push    esi
.text:01A5D8D6                  mov     esi, [eax]
.text:01A5D8D8                  xor     esi, edx
.text:01A5D8DA                  and     esi, 0F0F0F0F0h
.text:01A5D8E0                  xor     [eax], esi
.text:01A5D8E2                  xor     edx, esi
.text:01A5D8E4                  ror     edx, 14h
.text:01A5D8E7                  mov     [ecx], edx
.text:01A5D8E9                  push    edi
.text:01A5D8EA                  mov     edi, [eax]
.text:01A5D8EC                  xor     edx, edi
.text:01A5D8EE                  and     edx, 0FFFF0000h
.text:01A5D8F4                  mov     esi, edx
.text:01A5D8F6                  xor     [eax], esi
.text:01A5D8F8                  mov     edx, [ecx]
.text:01A5D8FA                  mov     edi, [eax]
.text:01A5D8FC                  xor     edx, esi
.text:01A5D8FE                  ror     edx, 12h
.text:01A5D901                  mov     [ecx], edx
.text:01A5D903                  mov     esi, [ecx]
.text:01A5D905                  xor     edx, edi
.text:01A5D907                  and     edx, 33333333h
.text:01A5D90D                  xor     [eax], edx
.text:01A5D90F                  mov     edi, [eax]
.text:01A5D911                  xor     esi, edx
```

```
static inline void IPERM(word32 &left, word32 &right)
{
        word32 work;

        right = rotlFixed(right, 4U);
        work = (left ^ right) & 0xf0f0f0f0;
        left ^= work;
        right = rotrFixed(right^work, 20U);
        work = (left ^ right) & 0xffff0000;
        left ^= work;
        right = rotrFixed(right^work, 18U);
        work = (left ^ right) & 0x33333333;
        left ^= work;
        right = rotrFixed(right^work, 6U);
        work = (left ^ right) & 0x00ff00ff;
        left ^= work;
        right = rotlFixed(right^work, 9U);
        work = (left ^ right) & 0xaaaaaaaa;
        left = rotlFixed(left^work, 1U);
        right ^= work;
}
```

Localizamos el resto de las funciones. La que más nos interesa es la que cifra y descifra los bloques:

```
CryptoPP_DES_EDE2_Base_ProcessAndXorBlock proc near
                                        ; DATA XREF: .rdat
                                        ; .rdata:01AA9918↓

var_4           = dword ptr -4
arg_0           = dword ptr  4
arg_4           = dword ptr  8
arg_8           = dword ptr  0Ch

                push    ecx
                mov     eax, [esp+4+arg_0]
                mov     edx, [eax+4]
                push    esi
                mov     esi, ecx
                mov     ecx, [eax]
                bswap   ecx
                mov     [esp+8+var_4], ecx
                bswap   edx
                push    edi
                lea     ecx, [esp+0Ch+arg_0]
                lea     eax, [esp+0Ch+var_4]
                mov     [esp+0Ch+arg_0], edx
                call    CryptoPP_RawDes_IPERM
                mov     eax, ecx
                push    eax
                lea     ecx, [esp+10h+var_4]
                push    ecx
                lea     edi, [esi+10h]
```

También vamos a localizar la parte donde envia y recibe Comandos-Respuestas a la tarjeta para poder seguir el log que vamos a hacer después:



Ya tenemos localizadas todas las partes que nos interesan. **Con el script para ida en python que viene a continuación pondremos breakpoints en cada parte interesante (en la entrada de la función de cifrado / descifrado para capturar los datos antes de pasar por el algoritmo; en la salida de la misma función para capturar los datos trás ser aplicado el algoritmo; y antes y despúes de llamar a SCardTransmit para capturar los datos enviados a la tarjeta y la respuesta recibida), y en el hook de breakpoints leeremos la zona de memoria donde están los datos enviados a / recibidos desde la tarjeta, y los datos cifrados y su correspondiente descifrado. El script también se guardará una lista de pares de datos cifrados y su correspondiente descifrado para luego aplicarlo a los datos capturados en SCardTransmit y poder ver bien lo que la dll está leyendo de la tarjeta.**

```
---------<<<RESPONSE<<<----------
87 69 01 30 18 0C 12 4B 70 72 69 76 41 75 74 65
6E 74 69 63 61 63 69 6F 6E 03 02 06 C0 30 2E 04
1D 41 30 36 33 37 43 31 31 41 37 38 30 42 33 31
32 30 31 30 30 31 31 38 31 31 30 35 30 37 03 03
06 30 00 01 01 FF 03 02 03 B8 02 01 01 A1 15 30
13 30 0D 04 04 3F 11 01 01 02 01 00 80 02 03 B0
02 02 08 00 80 00 00 00 00 00 00 99 02 90 00 8E
04 2E E9 32 43 90 00
.i.0...KprivAutenticacion....0..
.[redacted]..
.0.............0.0...?.........
...................2C..
------------------------------------------------
---------<<<CRYPTED RESPONSE<<<----------
87 69 01 CE CB 70 A8 72 37 FF B2 49 D8 E1 0C F8
59 F4 D2 58 7C 22 68 2F D5 8F 3C 30 8F 31 95 CD
0A 0C 58 01 D8 8B B5 94 D9 93 A6 A8 83 A0 BD 57
F2 64 0C E1 C4 5B 61 C4 2E 8E 3B E7 1A 8E BB F5
D3 D9 CF 2E FA 4B F7 7B C2 DA AA 1F CF 8C 4B 3B
CC 48 47 B0 DA 43 60 2A BD 65 57 AC D0 01 95 0F
8C 57 28 9F 72 03 BD 63 C4 D5 9A 99 02 90 00 8E
04 2E E9 32 43 90 00
.i...p.r7..I....Y..X|"h/..<0.1..
..X.............W.d...[a...;.....
.....K.{......K;.HG..C`*.eW.....
.W(.r..c...........2C..
------------------------------------------------
\\\\\\ProcessAndXorBlock\\\\\\
CE CB 70 A8 72 37 FF B2
..p.r7..
------------------------------------------------
//////ProcessAndXorBlock//////
30 18 0C 12 4B 70 72 69
0...Kpri
```

El script para hookear en las funciones de cifrado:

```
from idaapi import *
import ctypes
import struct

#################################################################################################################

def writelogitem(f,e):
  f.write("\n----------------------------------------------------------------------\n")

  f.write(e[0]+"\n")

  e=e[1]

  i=0
  for ee in e:
    f.write(("%02X " % ee))
    i+=1
    if i%16==0:
      f.write("\n")

  f.write("\n")
```

```
  i=0
  for ee in e:
    try:
      if ee>=0x20 and ee<0x7e:
        f.write(chr(ee))
      else:
        f.write(".")
    except:
      pass
    i+=1
    if i%32==0:
      f.write("\n")


#####################################################################################################################

def hex2ascii(s):
  l=s.split(" ")[5:]
  l2=[]
  rs=""
  for e in l:
    #if (int(e,16)>=0x61 and int(e,16)<0x7a) or (int(e,16)>=0x41 and int(e,16)<0x5a):
    if (int(e,16)>=0x20 and int(e,16)<0x7e):
      rs+=chr(int(e,16))
    else:
      rs+="?"
  return rs

#####################################################################################################################

def readmem(ea,sz):
    global processHandle
    #buffer = ctypes.c_char_p("_"*sz)
    buffer = ctypes.create_string_buffer(sz)
    bytesRead = ctypes.c_ulong(0)
    bufferSize =  sz
    ctypes.windll.kernel32.ReadProcessMemory(processHandle, ea, buffer, bufferSize, ctypes.byref(bytesRead))
    arr=[]
    for i in range(0,bytesRead.value):
      arr.append(ord(buffer[i]))
    return arr

#####################################################################################################################

def readdword(ea):
    global processHandle
    #buffer = ctypes.c_char_p("_"*4)
    buffer = ctypes.create_string_buffer(4)
    bytesRead = ctypes.c_ulong(0)
    bufferSize =  4
    ctypes.windll.kernel32.ReadProcessMemory(processHandle, ea, buffer, bufferSize, ctypes.byref(bytesRead))
    return struct.unpack("L",buffer.raw)[0]

#####################################################################################################################

class MyDbgHook(DBG_Hooks):

    def dbg_process_start(self, pid, tid, ea, name, base, size):
        global processHandle

        processHandle = ctypes.windll.kernel32.OpenProcess(0x1F0FFF, 0, pid)

        print "process handle!"
        print processHandle

        return 0

    def dbg_process_attach(self, pid, tid, ea, name, base, size):
        print "process attach"
        return self.dbg_process_start(pid, tid, ea, name, base, size)

    def dbg_process_exit(self, pid, tid, ea, code):

        global logs
        global logstypes
        global ProcessAndXorBlockInputOutput

        duplogs=[]
```

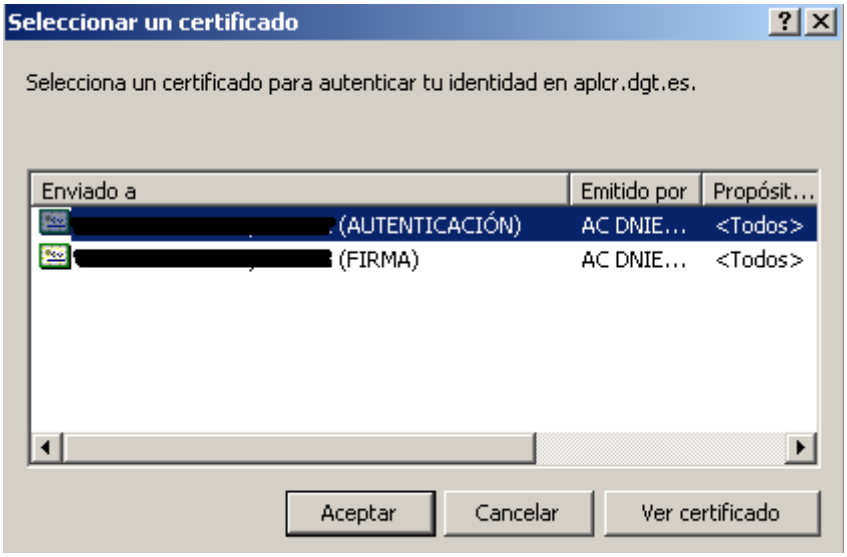
```
    for e in logs:
      dupe=[]
      for ee in e[1]:
        dupe.append(ee)
      tmparr=[]
      tmparr.append(e[0])
      tmparr.append(dupe)
      duplogs.append(tmparr)

    for e in ProcessAndXorBlockInputOutput:

      tempe=""
      for ee in e[0]:
        if ee>=0x20 and ee<0x7e:
          tempe+=chr(ee)

      if "Mast" not in tempe and\
        "File" not in tempe and\
        "cPKC" not in tempe and\
        "Cert" not in tempe and\
        "Autentic" not in tempe and\
        "acion" not in tempe and\
        "37C11A78" not in tempe and\
        "0B312010" not in tempe and\
        "01181105" not in tempe and\
        "ES1" not in tempe:
        nl=0
        for l in logs:
          if l[0]=="---------<<<RESPONSE<<<----------":
            i = 0
            while i + 8 <= len(l[1]):
              if l[1][i:i+8]==e[0]:
                l[1][i]=e[1][0]
                l[1][i+1]=e[1][1]
                l[1][i+2]=e[1][2]
                l[1][i+3]=e[1][3]
                l[1][i+4]=e[1][4]
                l[1][i+5]=e[1][5]
                l[1][i+6]=e[1][6]
                l[1][i+7]=e[1][7]
                logs[nl]=l
                break
              else:
                i+=1
          nl+=1


    f=open("c:\\logs.txt","w")

    nl=0

    for l in logs:
      writelogitem(f,l)
      if l[0]=="---------<<<RESPONSE<<<----------":
        duplogs[nl][0]="---------<<<CRYPTED RESPONSE<<<----------"
        writelogitem(f,duplogs[nl])
      nl+=1

    return 0

def dbg_process_detach(self, pid, tid, ea):
  return self.dbg_process_exit(pid, tid, ea, 0)

def dbg_library_load(self, pid, tid, ea, name, base, size):
    print "loaded:"+name
    return 0

def dbg_bpt(self, tid, ea):

    global logs
    global logstypes
    global LastProcessAndXorBlockInput
    global ProcessAndXorBlockInputOutput
    global LastRecvLenPtr
    global LastRecvBuffer

    arr=[]

    if ea==0x1A7F4BD:
      LastRecvLenPtr = GetRegValue("ECX")
```

```
        if ea==0x1A7F4C9:
          LastRecvBuffer = GetRegValue("EDX")

        if ea==0x1A5DF95: #ea==0x1A5E285 or
          arr.append("\\\\\\\\\\\\\\ProcessAndXorBlock\\\\\\\\\\\\\\")
          r = GetRegValue("EAX")
          LastProcessAndXorBlockInput=readmem(r,8)
          arr.append(LastProcessAndXorBlockInput)
          logs.append(arr)

        if ea==0x1A5E02E or ea==0x1A5E03E: #ea==0x1A5E306 or ea==0x1A5E2F6 or
          arr.append("//////ProcessAndXorBlock//////")
          r = GetRegValue("ECX")
          temp=[]
          temp.append(LastProcessAndXorBlockInput)
          temp.append(readmem(r,8))
          ProcessAndXorBlockInputOutput.append(temp)
          arr.append(temp[1])
          logs.append(arr)

        if ea==0x1A7F4CF:
          arr.append("--------->>>COMMAND>>>----------")
          r1 = GetRegValue("EDI")
          r2 = GetRegValue("EBP")
          arr.append(readmem(r1,r2))
          logs.append(arr)

        if ea==0x1A7F4D7:
          arr.append("---------<<<RESPONSE<<<----------")
          len=readdword(LastRecvLenPtr)
          arr.append(readmem(LastRecvBuffer,len))
          logs.append(arr)

        continue_process()
        return 0

    def dbg_trace(self, tid, ea):
        return 0

    def dbg_step_into(self):
        return 0

    def dbg_step_over(self):
        return 0
try:
    if debughook:
        print "Removing previous hook ..."
        debughook.unhook()
except:
    pass

#################################################################################################################################

# Install the debug hook
debughook = MyDbgHook()
debughook.hook()
debughook.steps = 0

logstypes=[]
logs=[]

LastRecvLenPtr=None
LastRecvBuffer=None
LastProcessAndXorBlockInput=None
ProcessAndXorBlockInputOutput=[]

processHandle=None

#SCardTransmit
add_bpt(0x1A7F4CF,0,BPT_SOFT)
enable_bpt(0x1A7F4CF,True)

'''
#CryptoPP_DES_Base_ProcessAndXorBlock (entrando)
add_bpt(0x1A5E285,0,BPT_SOFT)
enable_bpt(0x1A5E285,True)

#CryptoPP_DES_Base_ProcessAndXorBlock (saliendo)
add_bpt(0x1A5E306,0,BPT_SOFT)
```

```
enable_bpt(0x1A5E306,True)

#CryptoPP_DES_Base_ProcessAndXorBlock (saliendo)
add_bpt(0x1A5E2F6,0,BPT_SOFT)
enable_bpt(0x1A5E2F6,True)
'''

#CryptoPP_DES_EDE2_Base_ProcessAndXorBlock (entrando)
add_bpt(0x1A5DF95,0,BPT_SOFT)
enable_bpt(0x1A5DF95,True)

#CryptoPP_DES_EDE2_Base_ProcessAndXorBlock (saliendo)
add_bpt(0x1A5E02E,0,BPT_SOFT)
enable_bpt(0x1A5E02E,True)

#CryptoPP_DES_EDE2_Base_ProcessAndXorBlock (saliendo)
add_bpt(0x1A5E03E,0,BPT_SOFT)
enable_bpt(0x1A5E03E,True)

#Keep last recv len ptr
add_bpt(0x1A7F4BD,0,BPT_SOFT)
enable_bpt(0x1A7F4BD,True)

#Keep last recv buffer
add_bpt(0x1A7F4C9,0,BPT_SOFT)
enable_bpt(0x1A7F4C9,True)

#Keep response
add_bpt(0x1A7F4D7,0,BPT_SOFT)
enable_bpt(0x1A7F4D7,True)

# Start debugging
run_requests()
```

Análisis de la captura:

Con todo listo nos ponemos a depurar la dll (con internet explorer como host) conectando a la página de la dgt para consultar los puntos usando el DNIe (https://aplcr.dgt.es/WEB_COPACI/certificado/verSaldoPuntosCert.faces):



A continuación los EFs y DFs accedidos en la tarjeta:

Fichero Master.File -> cPKCS-15 -> 0x160:

.i.0...KprivAutenticacion....0..
.123456789A123456789A123456789.. -> ID clave autenticación
.0.....0.0...?.....
.....2C..

.i.0...KprivFirmaDigital....0...
123456789A123456789A12345678A... -> ID clave firmado
0@.....0.0...?.....
.....<0....

Fichero Master.File -> cPKCS-15 -> 0x460:

Un trozo:

.....0...CertAutenticacion....0. -> Trozo certificado X.509 autenticación
..123456789A123456789A123456789.
..0..0...`.p.....0w1.0...U...
.ES1.0...U....11111111K1.0...U..
..APELLID1.0...U.*..PERICO110/..
U...(APELLID APELLID, PERICO (AU
TENTICACI..N).^0\1.0...U....ES1(
0&..U....DIRECCION GENERAL DE...
LA POLICIA1.0...U....DNIE1.0...U...
.AC DNIE 003..D...../.....

Otro trozo:

.....0...CertFirmaDigital....0.. -> Trozo certificado X.509 firmado
.123456789A123456789A12345678A..
.0..0...`.p.....0n1.0...U....
ES1.0...U....11111111K1.0...U...
.APELLID1.0...U.*..PERICO1(0&..U
....APELLID APELLID, PERICO(FIR
MA).^0\1.0...U....ES1(0&..U....D
IRECCION GENERAL DE LA POLICI.u.
a.A1.0....
.).U....DNIE1.0...U....AC DNIE 0
03..D.....U<.b..

Otro trozo:

.....0...CertCAIntermediaDGP...@ -> Trozo certificado X.509 CA Intermedia
0...S0637C11A780B312010011811050
7...0..0...`ap.....0\1.0...U.
...ES1(0&..U....DIRECCION GENERA
L DE LA POLICIA1.0...U....DNIE1.
0...U....AC DNIE 003._0]1.0...U.
...ES1(0&..U....DIRECCION GENERA
L DE LA POLICIA1.0...U....DNIY..
a.
.).U....AC RAIZ DNIE..|l..p...D.
.n..H.....+l.t..

Fichero Master.File -> DNIe.Priv -> 0x470:

<<Bloque de datos sin interpretar>>

Fichero Master.File -> DNIe.Priv -> 0x570:

<<Bloque de datos sin interpretar>>

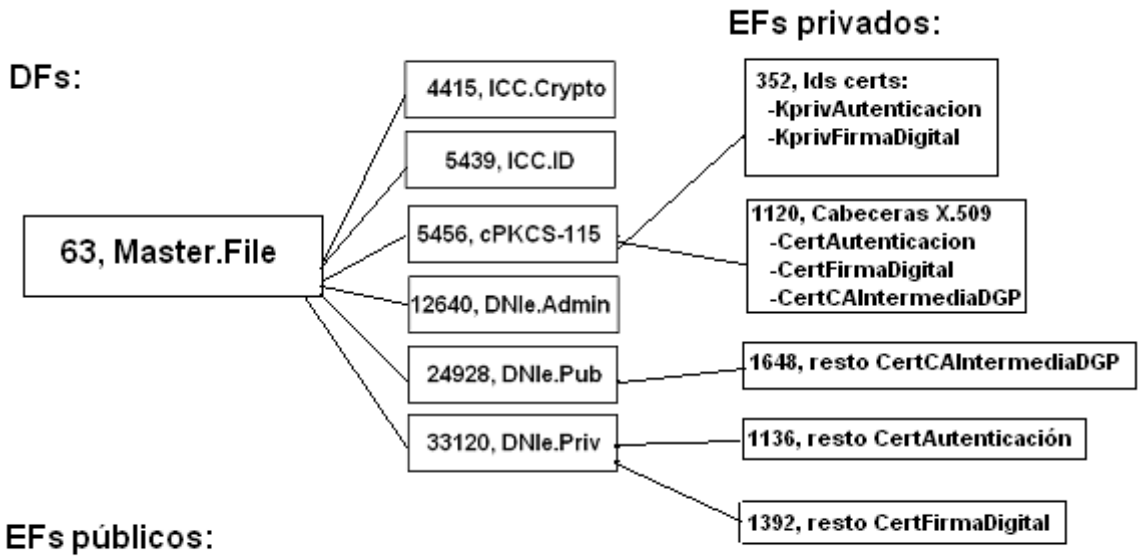
Fichero Master.File -> DNIe.Pub -> 0x670:

<<Bloque de datos sin interpretar>>

Los ficheros 0x470, 0x570 que cuelgan de DNIe.Priv, y el fichero 0x670, que cuelga de DNIe.Pub, creo que contienen el resto de los datos para los certificados de autenticación, de firmado y el de la CA intermedia, respectivamente.

7.3. RESULTADOS

7.3.1. PARTE DE LA ESTRUCTURA DE FICHEROS INTERNA DEL DNIE QUE HEMOS PODIDO COMPROBAR



EFs públicos:

815, Versión

1536, IDESP

8032, Certificado
CN = AC COMPONENTES 001
OU = FNMT
OU = DNIE
O = DIRECCION GENERAL DE LA POLICIA
C = ES

8288, Certificado
CN = 00000000657352444960006
OU = AC RAIZ COMPONENTES
OU = DNIE
O = DIRECCION GENERAL DE LA POLICIA
C = ES

43104, otros datos

A parte de lo anterior cuando hicimos el recorrido de ficheros sin habernos validado, encontramos cuatro ids que la tarjeta nos respondió que existían 1280, 4608, 8448 y 41312, pero que no pudimos leer el contenido. En las capturas de accesos a la parte privada no se accedía a estos ids por lo que no hemos podido ver el contenido (Queda pendiente implementar el recorrido de todos los ids (0-65535) de la tarjeta después de habernos validado).

7.3.2. CONCLUSIÓN

Hemos podido explorar gran parte de la estructura interna del DNI electrónico, aunque todavía se nos han quedado algunas partes pendientes para las cuales haría falta implementar el recorrido post-validación del pin de todos los ficheros por id (0-65535).

Según la web del DNIE existe otra zona, llamada zona de seguridad:

"ZONA DE SEGURIDAD: Accesible en lectura por el ciudadano, en los Puntos de Actualización del DNIe.

- *Datos de filiación del ciudadano (los mismos que están en el soporte físico).*
- *Imagen de la fotografía.*
- *Imagen de la firma manuscrita."*

Estaría bien comprobar si realmente los ids donde se almacenan estos datos son solamente accesibles desde los Puntos de Actualización en las comisarias, o simplemente es que el software que distribuyen no permite recuperar esta información (pero sí que está accesible, una vez validados). Como hemos visto en los logs, alguna de la información que aparece en el soporte físico (nombre y apellidos, número de DNI, IDESP) puede deducirse de los datos accesibles.

En las pruebas que hemos hecho hemos visto que no es muy complicado hookear en las funciones principales de cifrado y descifrado DES en la dll que se carga con los navegadores. Cualquier troyano podría facilmente inyectarse en Internet Explorer, buscar las funciones de cifrado DES y hookear en ellas para logear todos los datos intercambiados descifrados (entre ellos, datos que podrían ser comprometidos como claves públicas de autenticación y firmado, nombre y número de dni, idesp, etc...) sin necesidad de conocer el pin ni realizar ninguna implementación complicada.

El pin también sería sencillo de capturar en texto plano para un troyano con estos hooks, habría que esperar el comando VERIFY (0C 20 …) y mirar el penúltimo dato que se intentó cifrar: