**CHUID Parsing**

To parse a text file that contains the CHUID in one long delimited string of ASCII hex characters, use **txtchuid.pl *<file>*** where ***<file>*** is the text file. The result will be a group of **.bin** files whose name corresponds to the elements in the CHUID. Supported formats are:

* AABBCCDDEEFF1122AABB
* AA BB CC DD EE FF 11 22 AA BB
* AA:BB:CC:DD:EE:FF:11:11:AA:BB

Or, to parse a binary file that contains the CHUID, use **binchuid.pl *<file>*** where ***<file>*** is the text file. The result will be a group of **.bin** files whose name corresponds to the elements in the CHUID.

You can use **openssl asn1parse -inform der -in Issuer\_Signature.bin** to view the CHUID signature and even extract the CHUID signature certificate and in the example below:

openssl asn1parse -inform der -in Issuer\_Signature.bin | more

0:d=0 hl=4 l=2410 cons: SEQUENCE

4:d=1 hl=2 l= 9 prim: OBJECT :pkcs7-signedData

15:d=1 hl=4 l=2395 cons: cont [ 0 ]

19:d=2 hl=4 l=2391 cons: SEQUENCE

23:d=3 hl=2 l= 1 prim: INTEGER :03

26:d=3 hl=2 l= 15 cons: SET

28:d=4 hl=2 l= 13 cons: SEQUENCE

30:d=5 hl=2 l= 9 prim: OBJECT :sha256

41:d=5 hl=2 l= 0 prim: NULL

43:d=3 hl=2 l= 10 cons: SEQUENCE

45:d=4 hl=2 l= 8 prim: OBJECT :2.16.840.1.101.3.6.1

55:d=3 hl=4 l=1704 cons: cont [ 0 ]

59:d=4 hl=4 l=1700 cons: SEQUENCE

63:d=5 hl=4 l=1164 cons: SEQUENCE

67:d=6 hl=2 l= 3 cons: cont [ 0 ]

69:d=7 hl=2 l= 1 prim: INTEGER :02

72:d=6 hl=2 l= 18 prim: INTEGER :11216D464D266A183951872206765B2E9455

92:d=6 hl=2 l= 13 cons: SEQUENCE

94:d=7 hl=2 l= 9 prim: OBJECT :sha256WithRSAEncryption

105:d=7 hl=2 l= 0 prim: NULL

107:d=6 hl=3 l= 136 cons: SEQUENCE

110:d=7 hl=2 l= 11 cons: SET

112:d=8 hl=2 l= 9 cons: SEQUENCE

114:d=9 hl=2 l= 3 prim: OBJECT :countryName

119:d=9 hl=2 l= 2 prim: PRINTABLESTRING :US

123:d=7 hl=2 l= 39 cons: SET

The other CHUID elements are exported to files in binary form, and include:

* ***<file>.*FASC-N.bin**
* ***<file>*.DUNS.bin** (optional, and may not be produced)
* ***<file>*.GUID.bin**
* ***<file>*.Expiration\_Date.bin**

To view these files, use a command such as **od -tx1 <file>.**

A ***<file>.*sha-256** file is also created, and contains the SHA-256 of the elements in the CHUID as per SP 800-73-4, Section 3.1.2.

The signing certificate begins at the second sequence after the **id-PIV-CHUIDSecurityObject** (OID = 2.16.840.1.101.3.6.1). So, to extract it you can do this:

dd if=Issuer\_Signature.bin bs=1 skip=59 of=chuid.cer

openssl x509 -inform der -in chuid.cer -text

**Converting from Text to Binary**

To simply take ANY text file containing an ASCII Hex representation of ANY object of interest (fingerprints, facial images, certs), use **text2bin.pl** ***<file>*** where ***<file>*** is the text file. The result will be a file with the extension **.bin** appended to it. Supported formats are:

* AABBCCDDEEFF1122AABB
* AA BB CC DD EE FF 11 22 AA BB
* AA:BB:CC:DD:EE:FF:11:11:AA:BB

**Making a 200-bit Raw FASC-N Readable**

To convert a 200-bit TIG-SCEPACS FASC-N to readable format, use **tofascn.pl *<fascn>*** where ***<fascn>*** is the 200-bit form of the FASC-N.

To parse a fingerprint CBEFF that you’ve extracted out of a log file, first convert it to binary format using **text2bin.pl *<file>*** where ***<file>*** is the TLV data from the log file - one long string, actually. Then, use **finger.pl *<binfile>*** where ***<binfile>*** is the **.bin** file you produced with **text2bin.pl**. The result will be some informational output about the CBEFF, with the signature block written to a file whose name is the original text file name appended with **.cms**, which replaces the **.bin** extension. The **finger.pl** utility will also produce a CBEFF header file whose extension is **.cbeff**, and the FMR (finger minutiae record) whose extension is **.fmr**.

**Parsing Facial Image and Fingerprint Logs**

To parse a facial image CBEFF that you’ve extracted out of a log file, first convert it to binary format using **text2bin.pl *<file>*** where ***<file>*** is the TLV data from the log file - one long string, actually. The leading “BC” tag can be included in the text input and will automatically be handled, so that trimming isn’t always needed.

Use **facial.pl *<binfile>*** where ***<binfile>*** is the **.bin** file you produced with **text2bin.pl**. The result will be some informational output about the CBEFF, followed by the data in all three parts of the container in Base64 format. The **facial.pl** utility will produce a CBEFF header file whose extension is **.cbeffhdr** and an image file whose extension is **.jpg**. or **.jp2**, depending whether the Image Data Type field is 0 or 1 respectively, and a Cryptographic Message Syntax file (signature) whose extension is **.cms**.

Use **finger.pl *<binfile>*** where ***<binfile>*** is the **.bin** file you produced with **text2bin.pl**. The result will be some informational output about the CBEFF, followed by the data in all three parts of the container in Base64 format. The **finger.pl** utility will produce a CBEFF header file whose extension is **.cbeffhdr** and an fingerprint minutiae (FMR) file whose extension is **.fmr**, and a Cryptographic Message Syntax file (signature) whose extension is **.cms**.

You can use **openssl asn1parse -inform der -in *<cmsfile>*** where ***<cmsfile>*** is the **.cms** file you produced with **txtchuid.pl**, **binchuid.pl**, **finger.pl**, or **cbeff.pl**. This allows you to check whether the FASC-N and/or the UUID can be found.

If you want to run on Windows, use Cygwin with Perl installed. I have not tested these with ActiveState Perl. They should also run fine on Linux or OS-X.

/usr/local/bin binchuid.pl

/usr/local/bin/facial.pl

/usr/local/bin/finger.pl

/usr/local/bin/text2bin.pl

/usr/local/bin/tofascn.pl

/usr/local/bin/txtchuid.pl

/usr/local/lib/perl/lib/LogParser.pm

Perl prerequisite packages that you’ll need to pull from CPAN are:

Date::Calc

Digest::SHA

In order to build those packages on Cygwin, you’ll need a number of Cygwin packages such as gcc, autoconf, automake, and all of the MingGW stuff. When you install the Perl packages and the build fails, it gives a pretty good clue as to what Cygwin packages you need. You won’t have similar issues on Linux, since all of the builders are there.