Methods to Identify and Validate SAS®-Proprietary and AES Encrypted SAS® Data Sets

Thomas E. Billings, MUFG Union Bank, N.A., San Francisco, California

ABSTRACT

In a specialized context, this paper demonstrates how to read SAS® data sets and non-SAS files inside a DATA step using functions, and how to interrogate the SYSMSG() function to check for errors. Knowing how to do this is useful in a much wider context in the SAS ecosystem. The specific application addressed here: given a file that may or may not be a SAS data set, how can we check if it is an encrypted SAS data set? This is important for SAS server sites that rely on SAS-provided encryption to satisfy production requirements that "data at rest" must be encrypted. The task is trickier than expected as we show that encryption limits your ability to read (or even see) encrypted files in the SAS system. For files that follow SAS naming conventions, we first describe simple, predominantly manual methods to test for encryption – methods that produce error messages in the log when a file is encrypted. Then we explore a fully automated approach – appropriate for testing a large number of files - that does not produce error messages in the log. The methods described here are Base SAS and do not require metadata. Files that do not follow SAS naming conventions cannot be read as native SAS data sets by the SAS system. For these we read the file header and check for patterns. We end with recommendations for using these methods in a production program to spider the files in a server.

INTRODUCTION

The SAS® system supports encryption of native SAS data sets using 2 different encryption methods:

- a SAS proprietary method, and
- AES-256 encryption.

AES is short for Advanced Encryption Standard, and 256 refers to the use of a 256-bit key for encryption. Encryption can be applied at the data set level using data set options, via metadata, or in some contexts, using standard LIBNAMEs.

Some sites use metadata-bound, encrypted meta-LIBNAME SAS libraries to meet security requirements for encryption of permanent SAS data sets (i.e., "data at rest") in production environments. To maintain security, these sites need a way to identify and verify that a SAS data set is encrypted. Admin tools can be used to verify that datasets in certain directories are metadata-bound and encrypted, but SAS data sets can be written outside (or imported from outside) of the metadata environment.

This paper presents Base SAS, non-metadata methods to first identify if a file is or may be an encrypted SAS data set, and second, if encryption is suspected, multiple ways to validate or verify the type of encryption. The challenge here is the lack of encryption keys or passwords for some of the files; this significantly restricts what we can do with the files.

Validate encryption in this context means that there is positive evidence to support the claim that the file is encrypted; it does **not** mean that we decrypt the file as we won't have encryption keys or passwords for most files. We end with a discussion of issues in implementing this approach on a large server.

IDENTIFYING ENCRYPTION IN SAS FILES (.SAS7BDAT)

Data sets used for testing. For the analysis herein, a directory was created and test files containing artificial data were created (using a random number generator: SAS function RAND('UNIFORM')). Three data sets were created for each case:

- AES encrypted: files aes_enc_* where *=nq for an encryption key with no quotes, dq for double quotes, and sq for single quotes,
- SAS-proprietary encryption: files sas_prop_* where * = 1, 2, 3, and
- Unecrypted: files no_enc_* where * = 1, 2, 3.

PROC CONTENTS specifies if a SAS file is encrypted or not, but in order for the procedure to work the file must be opened and a password or encryption key must be supplied. In this case you need to know the answer to the question being asked; hence this PROC is not useful in this context.

PROC DATASETS can at least give you a list of SAS files in a directory, including encrypted files, even those for which you don't have the relevant passwords or encryption keys. The code below produces the standard file list from PROC DATASETS and also includes ODS statements to turn the TRACE option on/off, and to turn the OUTPUT option on/off to capture the file list in a SAS dataset for later use.

```
ods trace on;
ods output members=pd_list;

proc datasets lib=ct;
title "PROC DATASETS output for encryption test library";
quit;

ods output close;
ods trace off;

proc print data=pd_list;
title "File members from PROC DATASETS";
title2 "Captured via ODS";
run;
```

The above code produces the following printout (see next page).

PROC DATASETS standard output:

#	Name	Member Type	File Size	Last Modified
1	AES_ENC_DQ	DATA	128KB	11/06/2016 08:31
2	AES_ENC_NQ	DATA	128KB	11/06/2016 08:31
3	AES_ENC_SQ	DATA	128KB	11/06/2016 08:31
4	NO_ENC_1	DATA	128KB	11/06/2016 08:31
5	NO_ENC_2	DATA	128KB	11/06/2016 08:31
6	NO_ENC_3	DATA	128KB	11/06/2016 08:31
7	SAS_PROP_ENC_1	DATA	128KB	11/06/2016 08:31
8	SAS_PROP_ENC_2	DATA	128KB	11/06/2016 08:31
9	SAS_PROP_ENC_3	DATA	128KB	11/06/2016 08:31

Information from PROC DATASETS
Captured in a SAS file via ODS

Obs	Num	Name	MemType	FileSize	LastModified
1	1	AES_ENC_DQ	DATA	128KB	11/06/2016 08:31
2	2	AES_ENC_NQ	DATA	128KB	11/06/2016 08:31
3	3	AES_ENC_SQ	DATA	128KB	11/06/2016 08:31
4	4	NO_ENC_1	DATA	128KB	11/06/2016 08:31
5	5	NO_ENC_2	DATA	128KB	11/06/2016 08:31
6	6	NO_ENC_3	DATA	128KB	11/06/2016 08:31
7	7	SAS_PROP_ENC_1	DATA	128KB	11/06/2016 08:31
8	8	SAS_PROP_ENC_2	DATA	128KB	11/06/2016 08:31
9	9	SAS_PROP_ENC_3	DATA	128KB	11/06/2016 08:31

Unfortunately, PROC DATASETS has the same limitation as PROC CONTENTS: other than the basic file list above, PROC DATASETS needs the encryption key or password to open a SAS file for other uses.

Note that PROC DATASETS will list all SAS files in the LIBNAME directories. A single LIBNAME can point to multiple, concatenated directories, and in the work here a LIBNAME should point to one and only 1 directory, i.e., no concatenation.

SAS Dictionary Tables. These tables include a substantial amount of metadata, including an encryption variable in table “TABLES”. Let’s do a pull to check if this is useful. The code is:

```
proc sql;
      create table tparms as
            select libname, memname, memtype, encrypt
                  from dictionary.tables
                        where upcase(libname) = "CT";
```

```
quit;

proc print data=tparms;
title "File members from SAS dictionary tables";
run;
```

which produces the listing:

File members from SAS dictionary tables				
Obs	**libname**	**memname**	**memtype**	**encrypt**
1	CT	NO_ENC_1	DATA	NO
2	CT	NO_ENC_2	DATA	NO
3	CT	NO_ENC_3	DATA	NO
4	CT	SAS_PROP_ENC_1	DATA	
5	CT	SAS_PROP_ENC_2	DATA	
6	CT	SAS_PROP_ENC_3	DATA	

In the table above we observe:

- unencrypted data sets appear in the list as encrypt=NO,
- SAS proprietary data sets appear in the list as encrypt=missing, and
- AES encrypted data sets do not even appear in the dictionary tables listing!

Note the context here: we are using the dictionary tables on a single library that is not metadata-bound and that contains unencrypted data sets and both SAS-proprietary and AES-encrypted data sets. (Readers might want to verify/check if this issue is present on their site.) The results on a metadata-bound library might be different. However, we know that SAS data sets in an encrypted metadata-bound library will be encrypted, i.e., no need to test for encryption.

Comparing the file list from the dictionary tables with the list from PROC DATASETS, we can divide the superset of SAS data set names in the target LIBNAME/directory into 3 categories:

- unencrypted,
- believed to be SAS proprietary encrypted, and
- believed to be AES encrypted.

At this point we have tentatively identified certain SAS data sets as encrypted. However, the evidence supporting this classification is weak. The next section discusses additional methods to validate encryption.

VALIDATING ENCRYPTION IN SAS FILES (.SAS7BDAT)

Method 1: force an error. If a file is encrypted and you try to read it – in source code, with no password or encryption key supplied - it will throw an error when running in a non-interactive mode. (In an interactive mode, you may be prompted for the key or password, which you presumably don't have, which again throws an error.) PROC PRINT can be used for this purpose, with data set option (OBS=0) set to avoid any actual data printout. The log below illustrates this approach.

For an UNencrypted file, there is no error message and no data printed:

```
24          proc print data=ct.no_enc_1 (obs=0);
25          run;
```

```
NOTE: No observations in data set CT.NO_ENC_1.
```

For an AES encrypted file, the result is an error message:

```
27         proc print data=ct.aes_enc_nq (obs=0);
ERROR: Invalid ENCRYPTKEY value for CT.AES_ENC_NQ.DATA.
28         run;

NOTE: The SAS System stopped processing this step because of errors.
```

For a SAS proprietary encrypted file, the result is an error message:

```
30         proc print data=ct.sas_prop_enc_1 (obs=0);
ERROR: Invalid or missing READ password on member CT.SAS_PROP_ENC_1.DATA.
31         run;

NOTE: The SAS System stopped processing this step because of errors.
```

The error messages validate that the file is encrypted, and also report the type of encryption applied to the file.

This gives us **Method 1**: Use PROC PRINT with (OBS=0) as a data set option, to try to open the target file(s), per the code above. The error messages that result (if any) will report the encryption used, if any. This general approach has many variations and degrees of automation:

- If the number of files to be tested is small, the code can be written and run – i.e., files checked manually.
- To check a large number of files, a data/table driven approach can be used. Given a list of target data files:
  - Code to invoke PROC PRINT is generated in a macro or DATA step
  - ODS or PROC PRINTTO can be used to capture the resultant log in a text file
  - The code is run using %include (or other suitable method)
  - The log text file can then be opened and read in SAS, to check for error messages.

The approach of writing a SAS log to a file and then reading it for further analysis is a long-established practice in the SAS community, and there are numerous SAS User Group papers on the topic. The process of checking the error messages can be macroized: generate a PROC PRINT invocation and then interrogate the macro variable &SYSERRORTEXT after each PROC (rather than reading the log). This is trickier than one might expect, as the macro variable is set **only** when an error occurs, i.e., it presents the risk of false negatives when a no-error PROC PRINT invocation follows an invocation that generated an error message. Users choosing to macroize need to make sure their code checks for and detects false negatives.

Method 2: use DATA step functions. Method 1 is simple and is reasonable if you have a very small number of files to test, on a one-time basis. Because Method 1 relies on the program code generating error messages, it goes against the common practice that production jobs should not throw errors. Of course in this context there is good reason for the program to throw errors, but this may still be a concern for some users.

Method 1 was not automated here because there is an alternative method that does not send errors to the log, and is easier to automate: **Method 2**. In general, we can describe the method as follows.

Given: a LIBNAME pointing to a single directory (libref CT or ct in the code below) and a listing of target SAS data sets created by PROC DATASETS with ODS (per code above; file pd_list), the following basic processing is performed:

- Filter output file from PROC DATASETS to limit it to “DATA” member types
- In a new DATA step that creates a data set:
  - Read filtered file using SET, and for each file name in the input file:
  - If 1st row in input file, capture path to LIBNAME
  - Construct statement and invoke OPEN function to access the file
  - Use SYSMSG() function twice, 1st to get any error messages, and 2nd to clear
  - Check if error message returned in SYSMSG(), set encryption and error indicators
  - If target data set was opened successfully, CLOSE() it.

The code used is as follows.

```
data pd_list_data;
      set pd_list (keep=name memtype);

      if memtype = "DATA" then
            output;
run;

%let lbrf=CT;

%* libref for target LIBNAME;
data check;
      set pd_list_data;
      length lib_path $250. LFN $50. msg msg2 $100.;
      retain lib_path;
      enc_type = 0;
      unk_msg = 0;

      if (_N_ = 1) then
            lib_path = left(pathname("&lbrf."));
      LFN = cats("&lbrf..",name);
      ds = open(strip(LFN));
      msg=sysmsg();
      msg2 = sysmsg();

      if (index(upcase(msg),"INVALID ENCRYPTKEY VALUE") > 0) then
            enc_type = 1;

      if (index(upcase(msg),"INVALID OR MISSING READ PASSWORD") > 0) then
            enc_type = 2;

      if (not missing(strip(msg))) and (enc_type = 0) then
            unk_msg = 1;

      if (ds ne 0) then
            ds2=close(ds);
      drop msg2 LFN memtype;
      rename name=sas_file_name;
      output;
run;

proc print data=check;
title "Output file - result of opening SAS files";
run;
```

In this approach, error messages produced on attempted opens do not go to the log; instead they must be checked-for using SYSMSG() or relevant macro variable. The results from SYSMSG() may be one of the possible error messages if the file is encrypted; it may be missing if there is no error message, and it may return a message different from the anticipated ones. The code above creates indicator variables that identify these conditions.

Variable enc_type (for “encryption type”):

- = 0 unecrypted
- = 1 AES encryption
- = 2 SAS proprietary encryption

Variable unk_msg (for “unknown message”):

- = 0 expected system message (missing or “not encrypted” error message)
- = 1 other or not expected, i.e., “unknown” message; may merit investigation.

The resultant output SAS data set is shown below for the test files used here. It includes the path and SAS data set name for each file in the directory, along with the associated system error message returned (if any), plus the indicator variables enc_type and unk_msg, along with the return codes from invoking the OPEN() and CLOSE() functions. The return codes (variables ds, ds2) are omitted below to keep the table within the document margins.

Method 2 output file - result of opening SAS files

Obs	sas_file_name	lib_path	msg	enc_type	unk_msg
1	AES_ENC_DQ	/**redacted**/	ERROR: Invalid ENCRYPTKEY value for CT.AES_ENC_DQ.DATA.	1	0
2	AES_ENC_NQ	/**redacted**/	ERROR: Invalid ENCRYPTKEY value for CT.AES_ENC_NQ.DATA.	1	0
3	AES_ENC_SQ	/**redacted**/	ERROR: Invalid ENCRYPTKEY value for CT.AES_ENC_SQ.DATA.	1	0
4	NO_ENC_1	/**redacted**/		0	0
5	NO_ENC_2	/**redacted**/		0	0
6	NO_ENC_3	/**redacted**/		0	0
7	SAS_PROP_ENC_1	/**redacted**/	ERROR: Invalid or missing READ password on member CT.SAS_PROP_ENC_1.DATA.	2	0
8	SAS_PROP_ENC_2	/**redacted**/	ERROR: Invalid or missing READ password on member CT.SAS_PROP_ENC_2.DATA.	2	0
9	SAS_PROP_ENC_3	/**redacted**/	ERROR: Invalid or missing READ password on member CT.SAS_PROP_ENC_3.DATA.	2	0

Method 2 is recommended over Method 1 for the following reasons:

- Method 2 does not produce error messages in the log
- Method 1 will not work with the system option ERRORABEND specified or as default.
- Method 2 is simpler and it directly produces a SAS data set with the results

- Method 2 can easily be modified to process multiple directories/LIBNAMEs: use a sorted input file (BY libref or path) and incorporate calls to the LIBNAME function to setup and clear a LIBNAME for each target directory.

Constraints of Methods 1, 2. Given a single directory that may contain SAS files, one can use a version of method 1 or 2 to analyze the SAS files therein and determine their encryption status. However, PROC DATASETS and SAS dictionary tables provide information **only** on files in the directory that have the proper name/suffix (.sas7bdat for data sets). Alternative methods to address this limitation are the topic of a later section.

IMPLEMENTATION ISSUES: NESTED SUBDIRECTORIES

The preceding methods are constrained to operate in a single directory. How can we handle nested subdirectories? Also, could a user avoid their site encryption requirements by renaming files? What can be done for files that don't follow the SAS naming standards? Let's examine these topics, as follows.

Nested subdirectories. Hamilton (2012) and the SAS documentation (2016, link in reference section) present methods that use SAS DATA step functions to produce a list of files in a directory, including code that identifies subdirectories. All the files in a single subdirectory should be handled in a single DATA step. Also, a DATA step is required to check all the SAS files in each level of subdirectories found; this could be accomplished via macros to produce a DATA step per directory, or in a single DATA step if the file list is pre-sorted and first. and last.byvariable indicators are used.

The approach of Hamilton (2012) is not limited to SAS files; it will identify every file in the directory. This is needed as SAS files may be renamed in a directory using operating system commands for various reasons, e.g. "myfile.sas7bdat" might be renamed to:

- myfile.sas7bdat.backup or myfile.sas7bdat.bkp
- backup.myfile.sas7bdat.backup
- myfile.sas7bdat.old or myfile.sas7bdat.old

or other, similar names.

IMPLEMENTATION ISSUES: TESTING FILES THAT DO NOT FOLLOW SAS NAMING CONVENTIONS (NOT .SAS7BDDAT)

If a SAS data set is renamed using operating systems commands (as described above), the resultant file may be "invisible" to the SAS system. Here invisible means that the file does not appear in PROC DATASETS and it cannot be processed using PROCs as a normal SAS data set.

This is potentially problematic in a production environment where SAS is used to enforce production encryption and **all** (SAS) data sets need to be encrypted, including "old" or "backup" files on the server. The list of file names in a directory can be screened (using Perl regular expressions and/or SAS character functions) into categories:

1. SAS data set per SAS naming conventions
2. SAS file that is not a data set -- code, log, catalog, graphics files, etc.,
3. potential SAS file – does not follow naming conventions but may include "sas7bdat" as part of its name
4. other, probably non-SAS file.

The files of interest are those in category 3 and perhaps 4. We have a file that might be a SAS file, which might be any of: encrypted (AES or SAS proprietary) SAS data set; a regular, unencrypted SAS data set; or as per Monty Python, it might be "something completely different".

Standard methods for native SAS files don't work in this context. Because the target file does not follow SAS naming conventions, we cannot use the methods above to identify and validate SAS encryption. We cannot use PROC COPY (or other SAS method) to make a zero-row copy to work with, as the file is not recognized by the SAS system. You can rename a file that you own (to a standard SAS name) if you suspect it is a SAS data set and then test it, but this option is not available for files that you don't own, and in general renaming files in a production environment should be done only when there are compelling reasons.

There are examples of various hacks for SAS files that read the file header as raw, binary data – and not as a standard SAS data set. For files in category 3, 4, that is the only approach available. For this research, we examined the headers (first 400 characters) of the 9 test SAS data sets used for .sas7bdat files. Here are sample headers for each of the 3 file categories. Note the patterns highlighted below.

Raw header for file: aes_enc_nq.sas7bdat (SAS data set, AES encrypted)

```
????????????Âê?`³??Ï½???Ç1?????3"?33©®1©????????©????3"?33©®1©3©#3???? ©????
????SAS FILEAES_ENC_NQ
                        DATA    ?????RG?¸»ÚA?RG?¸»ÚA????? ÜÀ????? ÜÀ??©???©?©
??????????????9.0401M3Linux???????????2.6.32-642.3.1.e
??????????????x86_64???????????RG?Y?(?Y?(?Y?(????????AES     ?ù?Þ?????RG?¸
»ÚA????????????????????????????????????????????????
????
```

Raw header for file: sas_prop_enc_1.sas7bdat (SAS data set, SAS proprietary encryption)

```
????????????Âê?`³??Ï½???Ç1?????3"?33©®1©????????©????3"?33©®1©3©#3???? ©????
????SAS FILESAS_PROP_ENC_1
                        DATA    ????%°M?¸»ÚA%°M?¸»ÚA????? ÜÀ????? ÜÀ??©???©?©
??????????????9.0401M3Linux???????????2.6.32-642.3.1.e
??????????????x86_64???????????%°M?£b`0£b`0£b`0????????YZBENC01H??Þ????%°M?¸
»ÚA????????????????????????????????????????????????
????
```

Raw header for file: no_enc_1.sas7bdat (SAS data set, no encryption)

```
????????????Âê?`³??Ï½???Ç1?????3"?33©®1©????????©????3"?33©®1©3©#3???? ©????
????SAS FILENO_ENC_1
                        DATA    ????n3Q?¸»ÚAn3Q?¸»ÚA????? ÜÀ????? ÜÀ??©???©?©
??????????????9.0401M3Linux???????????2.6.32-642.3.1.e
??????????????x86_64???????????n3Q?"}>?"}>?"}>??????????????¼??Þ????n3Q?¸
»ÚA????????????????????????????????????????????????
????
```

The pattern, present in the first 400 characters of the header, appears to be:

- Locate the string "SAS FILE" (#1)
- If the "SAS FILE" string is present, capture the non-blank string (#2) that immediately follows. Check if this string is a potentially valid name per system option VALIDMEMNAME=COMPATIBLE **and** VALIDMEMNAME=EXTEND
- If string is possible valid member name, then scan the rest of the header for "DATA" (#3).
- If #3 is a yes, then scan remainder of header for "AES" (#4).

Results. Interrogating a file using this process, we infer that:

- The file MAY be a SAS file, AES encrypted, if all 4 conditions are satisfied
- The file MAY be a SAS file, unencrypted or SAS proprietary encrypted, if only the first 3 conditions are satisfied.

This process does not "prove" that a file is a SAS file; instead it identifies potential SAS data set files for further investigation and follow-up, re: encryption.

Another possible pattern test? Savvy readers will note this string in all header records above:

```
9.0401M3Linux
```

and that it might be automatically produced by suitable transformation logic on the automatic macro variables &SYSVLONG and &SYSSCPL. It is very tempting to test for this, as the more tests the better.

However, this is not recommended as over time, most sites use multiple versions of SAS, multiple operating systems (e.g., PC SAS on Windows and SAS on Linux/Unix or z/OS), and your users may import (via CEDA, see link in references for details) files from other systems that are different versions/operating systems. The process above could issue false negatives if a renamed SAS file was written under a SAS version/operating system that is not tested for explicitly.

Note that nonstandard SAS data file names (system option VALIDMEMNAME=EXTEND only) like, e.g., "ODD DATA" show up as ODD DATA in the header, i.e., not as 'ODD DATA'n. The headers above include numerous blanks, and limited tests suggest that yes, some of these are indeed blanks rather than unprintable binary characters. However we cannot assume that what we see as blank is actually blank in an arbitrary file being tested, and this makes it more challenging.

Constraint: for .sas7bdat files only. The pattern above was observed using native .sas7bdat files in SAS 9.4 Linux and Windows. Mainframe users should check the file headers on their system for compatibility with the algorithm. (z/OS supports Unix System Services; consider using that if appropriate.) The algorithm has not been developed for/tested on, and might not work for: transport format files, SPD engine files.

Prototype code to implement the pattern test. Prototype code to implement the pattern test is below. The code was used for testing the pattern algorithm and will need changes for productionization. Two DATA steps are used, the first to read the file headers, the second to check for the pattern.

Notes:

- The code should be modified to meet your requirements.
- The code includes error prints that can be removed if desired.
- The code highlighted below will need to be changed for production use: a / or \ is added to the path as a patch, for testing purposes. Also, the header should be tested for length in the 2nd DATA step before processing: if the header is a missing record or length <13 after STRIP is used to remove blanks, do not process.
- If you encounter "file not found" or other unexpected messages when using this logic, make sure the path/file name are the correct case. (Case sensitivity for file names can vary by system.)
- OPTIONS FORMCHAR is optional; suggested for portability.
- In the 2nd DATA step, we parse 32 characters after "SAS FILE" as a potential SAS data set name. This is trickier than it may seem. Both values of system option VALIDMEMNAME= must be supported. The code below is tentative and worked in testing for names that contain blanks. It may need modification for non-English language applications.

```
OPTIONS FORMCHAR="|----|+|---+=|-/\<>*";
libname MC "C:\SAS_more\dslib\";

%let fref=rawin;

data MC.header_as_data;

      set MC.sasfile_check (keep=sas_file_name lib_path);
      length fhead $400. fpname $250. fref $8. fmsg $200.;
      file print;
      fpname = cats(strip(lib_path),"\",strip(sas_file_name),".sas7bdat;
      *** Modify preceding line - sas7bdat files used for testing;
      ***for Linux testing, this code was used:;
      fpname =
cats(strip(lib_path),'/',strip(lowcase(sas_file_name)),".sas7bdat");
      put fpname=;
      fref="&fref.";
      rc_fn = filename(fref,strip(fpname));
      err_ind = 0;
      put rc_fn=;
      fmsg = sysmsg();
      put fmsg=;

      if (rc_fn = 0) then
            do;
                  rc_op = fopen(fref,"I",,"B");
                  put rc_op=;
                  fmsg = sysmsg();
                  put fmsg=;

                  if (rc_op ne 0) then
                        do;
                              rc_fr = fread(rc_op);
                              put rc_fr=;
                              call missing(fhead);

                              if (rc_fr = 0) then
                                    rc_fg = fget(rc_op,fhead,400);
                              rc_cl = fclose(rc_op);
                        end;

                  rc_fc = filename(strip(fref));
            end;
      else err_ind = 1;
      put @1 "Raw header for file";
      put @1 fhead $400.;
run;

data MC.potential_sasfile;
            set MC.header_as_data;
            length fhead fhead2 $400. lib_path $250. pt_sas_name $40.;

            sasfile_ind = 0;
            aes_ind = 0;
            c1 = index(fhead,"SAS FILE");
```

```
            if (c1 > 0) and (400-c1 ge 32) then
                  do;
                        pfn = substr(fhead,c1+8,32);
                        fc = substr(pfn,1,1);

                        if ((missing(fc)) or (fc = ".")) then
                              return;
                        pfn2 = compress(pfn,'|','kw');
                        pfn2 = strip(pfn);

                        if (missing(pfn2)) then
                              return;
                        c3 = length(strip(pfn2));

                        if (substr(pfn2,1,c3) ne substr(pfn,1,c3)) then
                              return;

                        if (nvalid(trim(pfn2),'V7') ne 1) then
                              do;
                                    if (findc(pfn2,'/\*?"<>|:') ne 0) then
                                          return;
                              end;

                        fhead2 = substr(fhead,c1+40,400-(c1+40));
                        LD = index(fhead2,"DATA");

                        if (LD > 0) then
                              do;
                                    sasfile_ind = 1;
                                    pt_sas_name = pfn2;
                              end;

                        fhead2 = substr(fhead2,LD+3,length(strip(fhead2))-
LD);

                        if (index(fhead2,"AES") > 0) then
                              aes_ind = 1;
                  end;

            keep lib_path pt_sas_name sasfile_ind aes_ind sas_file_name;
      run;
```

PRODUCTIONIZATION

This methods and logic above can be used to identify and validate - within limits - SAS files that are encrypted/unencrypted data sets, and also files that do not follow SAS naming conventions but may be native SAS data sets. For a site that wants to audit a SAS server to make sure **all** data sets are encrypted, more program logic is needed. The general, very high-level processing required for that is outlined as follows.

Given a set of target directories in a metadata environment:

1. Optional: check SAS metadata to identify all known encrypted directories and use as filter to identify potential unencrypted directories (see note below).
2. Spider through each (unencrypted) directory and get file lists:

3. Analyze file names and select files to be checked, i.e., exclude SAS non-data files and files known by their filenames to be non-data
4. For the remainder of the files: process using the methods in this paper, to identify actual/potential unencrypted SAS files.

Notes:

- The "/System/Secure Libraries" folder contains a list of metadata-bound libraries. This list can be manually accessed via the SAS Management Console or programmatically using PROC METADATA.
- As mentioned above, AES-encrypted files might not appear in the SAS dictionary tables. This may be a bug, and/or it may be fixed in a future release (or hot fix) of the SAS system. Keep this in mind when designing programs to check files. (Additional testing on this point at your SAS site is recommended.)
- The directory spidering process needs to run as a super user or privileged user with read access to all target directories.

Obviously, the outline above leaves out a lot of details. As textbooks sometimes say, the details here are left as exercises for the reader.

EPILOGUE & DISCLAIMER

The methods described herein were developed under SAS 9.4 and are ad-hoc. The logic here is prototype and tentative; it is **not guaranteed** in any way. It may need to be modified for non-English language sites, for future releases of the SAS system, and/or for other/future SAS products.

ACKNOWLEDGEMENTS

Thanks to Steve Holzworth of SAS Institute, Inc., for providing information on metadata-bound libraries. Any errors herein are solely the responsibility of the author.

APPENDIX 1:
BSD 2-CLAUSE COPYRIGHT LICENSE (OPEN SOURCE)

```
* All program code in this paper is released under a Berkeley Systems
  Distribution BSD-2-Clause license, an open-source license that permits
  free reuse and republication under conditions;
```

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
*/
```

REFERENCES

Note: all URLs quoted or cited herein were accessed in November 2016.

Hamilton J. (2012). Obtaining A List of Files In A Directory Using SAS® Functions. *Western Users of SAS Software Conference Proceedings.* URL: http://www.wuss.org/proceedings12/55.pdf

SAS Institute, Inc. (2016) online documentation:

- Cross-Environment Data Access (CEDA). *Moving and Accessing SAS(R) 9.4 Files, Second Edition.* URL: http://support.sas.com/documentation/cdl/en/movefile/67439/HTML/default/viewer.htm#p0c0l7xkprukh1n1ey1voicmgn6f.htm
- Example 2: List All Files within a Directory Including Subdirectories. *SAS(R) 9.4 Macro Language: Reference, Fourth Edition.* URL: https://support.sas.com/documentation/cdl/en/mcrolref/67912/HTML/default/viewer.htm#n0js70lrkxo6uvn1fl4a5aafnlgt.htm
- MVALID Function. *SAS(R) 9.4 Functions and CALL Routines: Reference, Fourth Edition*. URL: http://support.sas.com/documentation/cdl/en/lefunctionsref/67960/HTML/default/viewer.htm#n1auc6r6y41scsn1uc51pdcojmwy.htm
- Names in the SAS Language. *SAS(R) 9.4 Language Reference: Concepts, Fifth Edition.* URL: http://support.sas.com/documentation/cdl/en/lrcon/68089/HTML/default/viewer.htm#p18cdcs4v5wd2dn1q0x296d3qek6.htm
- NVALID Function. *SAS(R) 9.4 Functions and CALL Routines: Reference, Fourth Edition*. URL: http://support.sas.com/documentation/cdl/en/lefunctionsref/67960/HTML/default/viewer.htm#p19fp3i1f1hkomn12366grwnl8vt.htm

CONTACT INFORMATION

A list of the author's SAS-related papers, including URLs for free access, is available at:
http://www.sascommunity.org/wiki/Presentations:Tebillings_Papers_and_Presentations
or use this alternate short URL: http://goo.gl/uocYNc

Thomas E. Billings
MUFG Union Bank, N.A.
Basel II - Retail Credit BTMU
350 California St.; 6th floor
San Francisco, CA 94104

Phone: 415-273-2522
Email: tebillings@gmail.com