Keeping Passwords, AES Encryption Keys, and Other Sensitive Parameters Out of Source Code and Logs

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

ABSTRACT

The use of metadata-bound, encrypted libraries (with meta-LIBNAMEs) is one way to keep SAS® data set passwords and encryption keys out of source code and logs. However, that approach is constrained to the client-server metadata environment and data set/LIBNAME access, and at many sites requires SAS Admin support. This paper builds on previous work (Billings 2017A) and presents a simple, easy, secure, non-metadata method for keeping passwords, encryption keys and other sensitive/confidential single-string/token parameters out of the code and log. Compiled, encrypted function-style macros – written in a specific way - are used to supply the sensitive information. Compiled, encrypted macros are not as secure as one might expect; methods are described here to check for and mask sensitive parameters in the executable form of compiled macros. This approach was originally developed to supply (and protect) AES encryption keys and SAS-proprietary passwords for SAS data sets. However - and fortunately - this method can also be used to protect/conceal passwords & other sensitive parameters in a wide range of SAS statements (e.g., LIBNAME, FILENAME, ODS pdf passwords, parameters in PROC SQL CONNECT syntax, and Excel passwords in SAS code).

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. AES encryption is widely used and has been adopted as a standard by the U.S. and other governments. AES encryption is new in SAS 9.4 and is not available in earlier releases of the SAS system.

Encryption can be applied at the data set level using data set options and via metadata-bound meta-LIBNAMEs (or in some contexts, standard LIBNAMEs). To encrypt a file using data set options, use code like the following when creating the file.

For SAS proprietary encryption:

- `data mylib.myfile (encrypt=yes pw=password);`
- `data mylib.myfile (encrypt=yes write=password);`
- `data mylib.myfile (encrypt=yes alter=password);`
- `data mylib.myfile (encrypt=yes read=password);`

Using PW= is recommended as it sets ALTER=, WRITE=, and READ= passwords in a single option; it is also simpler. SAS proprietary encryption uses relatively weak passwords, i.e., the passwords must be a valid SAS name, maximum of 8 characters in length, and are case-insensitive.

For AES encryption (SAS 9.4+):

- `data mylib.myfile (encrypt=aes encryptkey=key);  /*no quotes*/`
- `data mylib.myfile (encrypt=aes encryptkey="key"); /*double quotes*/`
- `data mylib.myfile (encrypt=aes encryptkey='key'); /*single quotes*/`

The rules for what constitutes a valid key vary somewhat depending on whether the no quotes, double quotes, or single quotes format is used; consult the SAS documentation for details (check reference section for relevant URL).

To use an encrypted file in code, reference the file using the relevant data set option:

```
mylib.myfile (read=password)
mylib.myfile (encryptkey=*)   and * = one of: key, "key", 'key'
```

The requirement to supply the password or key with the data set name in code is problematic as best practice is to keep sensitive or confidential information out of source code and SAS logs.

Existing methods to keep passwords out of code/logs. One way to avoid showing the keys or passwords in code (and/or in the log) is to setup a meta-LIBNAME that is metadata bound and encrypted. If your SAS installation includes a metadata server and you are allowed to use PROC AUTHLIB, then you can do this directly; otherwise you will need SAS Admin assistance to set this up.

Base SAS includes a procedure that can replace a password with an encoded string (PROC PWENCODE). This keeps the actual password out of the code but the encoded string is usable as a proxy for the password in SAS, hence it provides only limited security. The encoded passwords from PROC PWENCODE have additional limitations, discussed below.

The SAS documentation provides suggestions for ways to keep passwords and encryption keys out of the log (but not out of code); see the references section for the relevant URL. Hemedinger (2010) also provides multiple strategies for keeping passwords out of source code.

This paper presents an alternative: simple & secure methods to keep sensitive/confidential parameters out of code/log that do not require metadata, meta-LIBNAMEs, authentication domains, or external authentication systems. The methods presented here have wider applications as well, including, e.g., sensitive parameters in LIBNAMEs to PCFILES server or databases, ODS system options, and other statements.

CHALLENGES AND CONSTRAINTS

Very important advice. When you create an AES-encrypted file in SAS, the SAS system issues this note of advice:

```
NOTE: If you lose or forget the ENCRYPTKEY, there will be no way to open the
file and recover the data.
```

The above note should be taken very seriously, because:

data loss caused by a missing encryption key could have a major and negative impact on your employer and/or your career.

Caution and due diligence should be exercised in keeping a secure record of keys used. Safeguarding the keys should be a critical part of data governance in enterprises.

Format of the key is important. As described above, the AES encryption key can be in one of 3 different formats – a plain string (no quotes), in double quotes, or in single quotes. One might think (naively) that if a string is used as key in one format, then that same key would work in the other formats, after adding or subtracting quotes. However, this is only partially true; the table below shows how a file saved with an AES key in 1 format can throw an error when a read is attempted using the same key in a different format:

SAS data set created using AES encryption key in format:	**Data set access method:**		
	No quotes	**Double quotes (")**	**Single quotes (')**
No quotes	Works	Throws error	Throws error
Double quotes (")	Throws error	Works	Works
Single quotes (')	Throws error	Works; macro quoting constraint	Works

The reason for the errors above is structural: an AES key that is valid in single or double quotes may throw a syntax error when used without quotes.

> **Note:** the encryption key string (by itself, no quotes) can be supplied via a macro/macro variable in the code for the no-quotes and double quotes formats, but not the single quotes option as single quotes prevent macro expansion. To clarify this point: if we create a file with the key "xyz" (double quotes) then we can supply the key as a macro/macro variable in any of the forms: 1) "&key." or "%key" so long as &key, %key resolve to xyz (no quotes), or 2) &key., %key so long as these resolve to "xyz", i.e., xyz embedded in double quotes.

Limited flexibility in specifying passwords or keys. A number of techniques that are sometimes used to mask or hide strings do **not** work as operands for PW=, ENCRYPTKEY=:

- SYMGET("macro_variable_name")
- RESOLVE('&macro_variable_name')
- Passwords or keys encoded using PROC PWENCODE

Other methods to hide the code might be N/A. System options NOMPRINT, NOSYMBOLGEN, NOMLOGIC can be specified to keep information out of the log (from macro-run code), but this approach can be cumbersome as it requires encapsulating code in macros and additional coding. This approach might not be feasible (or cost-effective) for many applications.

Additionally, the system options NOSOURCE, NOSOURCE2, and NONOTES may be needed to keep sensitive information out of the log. Sensitive parameters can be held in macro variables but the %LET statement will show in source code, even if the appropriate options are set to suppress the log. Also, sensitive data stored in global macro variables can be exposed by a simple %PUT _ALL_; statement after the fact (in the same SAS session).

AN ALTERNATE APPROACH:
FUNCTION-STYLE COMPILED, ENCRYPTED MACROS

The difficulties in keeping passwords and keys out of source code and the log are described above. However, there is a method that works: make the sensitive information, whether a plain text string with no quotes or a string denoted by single/double quotes, the output produced by a function-style compiled, encrypted macro. To make the macro secure, it should include tests on &SYSUSERID, i.e., the macro can be made to work for 1 and only 1 userid if desired.

Macros are needed for PW= and each of the 3 forms of ENCRYPTKEY=. The code below works; additional discussion follows the text. All code in this paper is released under the BSD 2-clause open source copyright license which allows free reuse under conditions; see Appendix 1 for details. The data sets used below contained artificial data, and the encryption key and password are also artificial hence can be public.

For SAS proprietary passwords:

```
%macro mypswd/ store secure;
%*macro mypswd;
      %local saspswd myuserid;
      %let myuserid=redacted;
      %let saspswd=X1Y_59KM;
      %if (&sysuserid. = &myuserid.) %then %superq(saspswd);
            %let saspswd=;
            %let myuserid=;
%mend;
```

For AES encryption key; key has no quotes:

```
%macro myaes1/ store secure;
%*macro myaes;
      %local EK myuserid;
      %let myuserid=redacted;
      %let
EK=XTPzMEd8pvIDBmlXKfjrwvNk0fAfTXiwmUNYniT0fOhKibzs3dEhVvCCQeyF8U2K;
      %if (&sysuserid. = &myuserid.) %then %superq(EK);
            %let EK=;
            %let myuserid=;
%mend;
```

For AES encryption key; key in double quotes:

```
%macro myaes2/ store secure;
%*macro myaes;
      %local EK myuserid;
      %let myuserid=redacted;
      %let
EK="XTPzMEd8pvIDBmlXKfjrwvNk0fAfTXiwmUNYniT0fOhKibzs3dEhVvCCQeyF8U2K";
      %if (&sysuserid. = &myuserid.) %then %unquote(&EK.);
            %let EK=;
            %let myuserid=;
%mend;
```

For AES encryption key; key in single quotes:

```
%macro myaes3/ store secure;
```

```
%*macro myaes;
      %local EK myuserid;
      %let myuserid=redacted;
      %let
EK='XTPzMEd8pvIDBmlXKfjrwvNk0fAfTXiwmUNYniT0fOhKibzs3dEhVvCCQeyF8U2K';
      %if (&sysuserid. = &myuserid.) %then %unquote(&EK.);
            %let EK=;
            %let myuserid=;
%mend;
```

Notes/details:

1. **Creating compiled macro libraries.** The macros need to be stored in a compiled, encrypted macro library, per the /STORE SECURE option on the %MACRO statement. Autocall libraries are not appropriate because they contain unencrypted source code.
2. **Using compiled macro libraries.** To use compiled macros, the system options MSTORED and SASMSTORE= must be specified; the latter specifies a libref or catref for the stored macro catalog/library. The macro library should be ACCESS=READONLY in LIBNAME statements in normal usage; the only exception being jobs that create compiled macros. Compiled macro libraries can be concatenated if necessary.
3. **Security: delete logs, change code after compilations.** The code used to compile the macros and associated log will contain sensitive information. These logs should be deleted and the macro source code modified to remove sensitive information, before the file gets copied to nightly backup or to a source code repository. A secure record of the passwords/keys should be kept, possibly offline.
4. **Security: the macro works for only 1 userid.** This code in the macro: `%if (&sysuserid. = &myuserid.) %then` limits the macro to work only for a single userid, and provides security. This can be modified to include other userids, or omitted. (Warning: If this check is omitted, then the macro is **not** secure!) If a list of userids must be supported, the code can be modified to accommodate multiple users, but take note: that induces a maintenance requirement.
5. **Security: complementary with PROC PWENCODE**. This method also works to deliver database passwords encoded using PROC PWENCODE, for additional security. However, recall that passwords/keys encoded with PWENCODE do not work for SAS data sets.
6. **Security: operating system and metadata permissions.** Access to the directory containing the executable form of the compiled macro can be restricted using operating system and/or SAS metadata permissions. Having multiple layers of security is a "best practice".
7. **Special characters and macro quoting.** The single/double quote macro versions here were tested using keys/passwords that had no special characters. Ideally, the macros should not throw an error because of special characters. To avoid this, for AES encryption keys, use the no quotes form for simplicity as it disallows special characters. If possible, avoid use of special characters in strings (passwords or userids) to be protected with these macros. If special characters must be supported, you may need to modify the macro code to use the appropriate macro quoting functions (e.g., %SUPERQ) and/or mask any unmatched single or double quotes or parentheses in strings defined using %STR or %NRSTR. For more information, see the References section for a link to SAS documentation on macro quoting functions.
8. **Function style macros: potentially fragile code.** To work correctly, function-style macros must return a character string with no semicolons. The code won't work correctly if:
   - A null statement (; only or blanks plus a ;) is inserted in the macro
   - Comments of the form: * comment ; are inserted in the code
9. **Constraints on macro names.** Testing during development found that macro names like %mymacro(num=1) will throw an error when used as SAS data set passwords or encryption keys; it seems that they do not parse correctly in some of the contexts tested. Suggestion: keep the macro names short (max. 8 characters including % character), and don't have any macro parameters. (Note, however, that longer macro names – with no arguments - did work with tests for relational database passwords.)
10. There should be 1 macro for each parameter.

11. For sample code to generate random AES keys, see Billings (2017A).

Constraint: compiled, encrypted macros are not secure. The use of compiled, encrypted macros does **not** guarantee security. Opening a compiled macro file in a text editor like notepad or vi reveals that some/many SAS macro statements are masked in the compiled file, but some SAS code may appear as plain text on the editor screen. In some cases, macro statements may be visible in the compiled file. This is a security issue if sensitive information is visible in the compiled macro file.

The macros above uses the method of encapsulating sensitive parameters in %LOCAL macro variables, with a %LET statement in the code, which is cleared after the macro runs. This method is described in Billings (2017A) and it appears to work when only 1 macro is compiled.

Adding an additional macro to the compile, a stub macro, is worth doing: in tests done for this paper, the code for the last macro (of n>1 macros) compiled may be visible in the executable macro file, but code for the earlier macros (compiled in the same run) was hidden. So if a stub macro is omitted or compiled before the last macro in a compilation, then sensitive information may be revealed in the executable file. If a stub macro is compiled last, its code may be exposed (and that does not matter) but the code for the preceding macros may be hidden.

In this context, a stub macro is very simple: a %MACRO statement with a name, %RETURN , and %MEND. Note: This information is observational and cannot be guaranteed. Always check the compiled macro executable file to make sure that sensitive information is not visible! If the hacks mentioned here do not work to hide sensitive information, then you may need to develop other hacks.

Of course if you are the **only** user who can access the compiled macro catalog file as enforced by operating system or metadata permissions, then you may be able to skip some of the security checks above.

The macros work and keep passwords/keys out of the code/logs. The macros above were compiled and the test code below was run successfully against data sets containing artificial data. Data sets test_1, 2, 3 are aes-encrypted files whose key has no quotes, double quotes, and single quotes respectively. Data set text_4 is encrypted using SAS proprietary encryption.

In the log below, the associated macro is invoked in the code in place of the sensitive parameters. Note that macro %myaes1 outputs a string without quotes and can be used successfully inside double quotes (3rd PROC PRINT below). The macros %myaes2, %myaes3 respectively output a string in double/single quotes, and %mypswd outputs a SAS proprietary [password.

```
25          options mprint symbolgen mlogic macrogen;

27          libname stdmcr "/redacted/" access=readonly;
NOTE: Libref STDMCR was successfully assigned as follows:
      Engine:        V9
      Physical Name: /redacted/
28          options mstored sasmstore=stdmcr;
29          options nocenter dtreset;

31          libname CT "/redacted/;
NOTE: Libref CT was successfully assigned as follows:
      Engine:        V9
      Physical Name: /redacted/

33          proc print data=ct.test_1 (encryptkey=%myaes1);
34          run;

NOTE: There were 100 observations read from the data set CT.TEST_1.
```

```
36          proc print data=ct.test_2 (encryptkey = %myaes2);
37          run;

NOTE: There were 100 observations read from the data set CT.TEST_2.

39          proc print data=ct.test_2 (encryptkey = "%myaes1");
40          run;

NOTE: There were 100 observations read from the data set CT.TEST_2.

42          proc print data=ct.test_3 (encryptkey = %myaes3);
43          run;

NOTE: There were 100 observations read from the data set CT.TEST_3.

45          proc print data=ct.test_4 (PW=%mypswd);
46          run;

NOTE: There were 100 observations read from the data set CT.TEST_4.
```

The test code/log above shows that the approach of using encrypted, compiled function-style macros (coded as described here) to provide the password/encryption key in data set options works. Furthermore, the values of the password/encryption key do **not** appear in the source code or log, despite the use of SAS system options that normally disclose the internal operations/code generated by macros. When the macro code includes the test on &sysuserid as done here, then the macro will not work for anyone but the authorized userid as &sysuserid is system-set and cannot be reset by a user.

Additionally, the common macro hacks:

```
%put %macro_name;
```

and/or

```
%let secret=%macro_name;
%put &secret.;
```

do **not** work with these compiled macros for userid(s) other than the authorized userid(s).

ADDITIONAL APPLICATIONS

The method works to assign parameters for the ODS pdf password system option statement:

```
options pdfsecurity=high  pdfpassword=(owner=%spuser open="%sppswd");
```

It also works with LIBNAME parameters, using macros with output that has no quotes, double quotes, and single-quotes:

```
LIBNAME ra_sch ORACLE  PATH="&path"  SCHEMA= %spschm  USER= %spuser PASSWORD=
"%sppswd" ;
```

These macros can be used in the Excel DDE FILENAME approach to keep Excel file passwords out of the log/code, but at the same time feeding them to DDE. (Test macros were used with some of the code of Hao (2013) and the output string verified by visual inspection.) The macros also work to supply passwords & userids for PROC SQL CONNECT syntax.

In the applications above, no sensitive parameters appeared in the code or log, despite the invocation of system options MPRINT, SYMBOLGEN, MLOGIC, and the obsolete MACROGEN. Given the preceding, it follows that the use of compiled, encrypted macros to supply sensitive strings is not limited to PW=, READ=, ALTER=, WRITE=, or ENCRYPTKEY= fields in data set options. Presumably it can be used wherever single-token strings need to be protected in SAS code.

IMPLEMENTATION ISSUES

Test first. ALWAYS test your macro before using it in code. To test the compiled macro, use code like the following, run under your userid:

```
%let myvar=%mymacro;
%put &myvar.;
```

Per the preceding section, the code above will **not** reveal your password if run under a userid other than yours. The macro code shown here includes a commented-out %*MACRO statement specifically to make it easy to create a version of the macro that is not stored/encrypted, for testing.

Possible compiled macro library access issues. If you are working in SAS Enterprise Guide and you compile a macro and then try to use it in the very same run, you may encounter access issues even if you issue LIBNAMEs to clear the relevant libref and then reassign it. When this happens, simply save the project/code, exit, then restart SAS Enterprise Guide and use the just-compiled library. For more options to try in this situation, see Hemedinger (2016).

Parsing errors were not an issue when testing these macros in development. They were an issue in the related and more complex macro in Billings (2017A). Anyway, if your macro tests show a correct string/token but it throws errors when run, you may need to do one or more of the following.

- Use different macro quoting functions; ones that work at compile time and/or others that work at run time.
- In the SAS code where the macro is invoked, insert spaces between keyword and =, i.e., keyword = %mymacro, or force spaces with %STR( ) if the code is part of a macro variable.
- If building a long string with multiple parameters, try assigning it to an intermediate variable and use that in the target statement (and be sure to clear the intermediate variable after use).

Suggestion: provide support for beginners. The macros are relatively simple (so long as you don't need to support special characters!) but it is also very easy to make mistakes. This paper is written for intermediate/advanced programmers who are comfortable with the nuances of macro code/quoting and compiling macro libraries. Beginners might benefit from additional support, i.e.:

- develop stored processes (and/or SAS Studio tasks) to create/manage a user compiled macro library,
- provide prototype code with instructions on how to fill in the blanks and compile the macros,
- a short training class which might be recorded and available for replay.

CONCLUSION

Billings (2017A) presents a **centralized** approach to securely assign LIBNAMEs to relational databases using a database parameter file. This paper presents a **decentralized** way to supply individual parameters (tokens) like passwords, encryption keys, userids, in a secure way, i.e., the strings/tokens do not appear in the source code or log, and cannot be exposed using known hacks. The approach here uses compiled, encrypted, function-style macros that supply the string/token only if the current userid matches the value of &sysuserid. Compiled, encrypted macros are not completely secure and steps are described to check for sensitive parameters in the executable form of compiled macros, with suggested mitigations. The method was developed to secure SAS proprietary passwords and AES encryption keys in data set options, but in fact is applicable to a much wider range of applications in SAS.

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 copyright license, an open-source license
  that permits free reuse and republication under conditions;

/*
Copyright (c) 2017, MUFG Union Bank, N.A.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

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.
*/
```

APPENDIX 2:
METHODS FOR HIDING SENSITIVE PARAMETERS IN SAS: SOURCE CODE & LOGS

This paper - referred to as Billings (2017B) below - and another related paper, Billings (2017A), provide non-metadata methods for keeping sensitive parameters out of source code and logs. The table below summarizes the methods developed in these 2 papers, and show how these methods fill in gaps – applications not supported by metadata.

Methods for Hiding Sensitive Parameters in SAS: Source Code & Logs

Type of sensitive information	Context	Metadata-based methods	Non-metadata methods
AES encryption keys, SAS proprietary passwords	Data set options, specialized LIBNAMEs	Metadata-bound libraries	Billings (2017B) macro method
DBMS userids, passwords	Accessing databases via LIBNAMEs	Meta-LIBNAMEs; Authentication domains	Billings (2017A, B) macro methods
DBMS userids, passwords	Accessing databases using CONNECT TO syntax	Authentication domains work, but this is not clear in SAS documentation. Alternate syntax: CONNECT USING libref.	Sherman & Carpenter (2009) method; Billings (2017B) method
Userids, passwords for most external file interfaces	FILENAMEs: ftp, sftp, URL, WebDAV, etc.	Authentication domains	Billings (2017B) macro method
Userids, passwords for files	Authentication domains NOT supported: LIBNAME for data engines PCFILES server, EXCEL, ACCESS; PROC IMPORT, EXPORT, etc.	N/A; use non-metadata method.	Billings (2017B) macro method
PDF, Excel passwords (output files)	Writing password-protected pdf, Excel files in SAS code	N/A; use non-metadata method	Billings (2017B) macro method

ACKNOWLEDGEMENTS

Thanks to Paul Sherman and Art Carpenter for their informative and interesting 2009 paper, which inspired this paper. Thanks to Chris Hemedinger of SAS Institute, Inc., for his blog postings and for information on authentication domains. Any errors herein are solely the responsibility of the author.

REFERENCES

Note: all URLs quoted or cited herein were accessed in May 2017.

Billings T (2017A). Secure Macro-Based Method to Assign LIBNAMEs for Databases. Paper to be submitted to 2017 *Western Users of SAS Software.*

Billings T (2017B). This paper; to be submitted to 2017 *Western Users of SAS Software.*

Hao, J (2013). SAS Automation - From Password Protected Excel Raw Data to Professional-Looking PowerPoint Report. *Midwest SAS Users Group Conference Proceedings.* URL: http://www.mwsug.org/proceedings/2013/FS/MWSUG-2013-FS04.pdf

Hemedinger C (2016). Tip: How to close all data sets in SAS Enterprise Guide. *SAS Blog: The SAS Dummy.* URL: http://blogs.sas.com/content/sasdummy/2016/10/18/close-data-sas-eg/

Hemedinger C (2010). Five strategies to eliminate passwords from your SAS programs. *SAS Blog: The SAS Dummy.* URL: http://blogs.sas.com/content/sasdummy/2010/11/23/five-strategies-to-eliminate-passwords-from-your-sas-programs/

SAS Institute, Inc. online documentation:

- Blotting Passwords and Encryption Key Values. *SAS® 9.4 Language Reference: Concepts, Fifth Edition. URL*:http://support.sas.com/documentation/cdl/en/lrcon/68089/HTML/default/viewer.htm#n0f79bcfsnsl82n117dahzoqrsdn.htm
- ENCRYPTKEY= Data Set Option. *SAS® 9.4 Data Set Options: Reference.* URL: https://support.sas.com/documentation/cdl/en/ledsoptsref/68025/HTML/default/viewer.htm#n0yzx049gh8pn3n1v7yrzagard3a.htm
- Macro Quoting. *SAS® 9.4 Macro Language: Reference, Fifth Edition.* URL: http://support.sas.com/documentation/cdl/en/mcrolref/69726/HTML/default/viewer.htm#p07u5itr1teq0dn1bx0lli1ri5dy.htm
- *SAS® 9.4 Guide to Metadata-Bound Libraries, Second Edition.* URL: http://support.sas.com/documentation/cdl/en/seclibag/66930/HTML/default/viewer.htm#titlepage.htm
- Usage Note 38204: Using the AUTHDOMAIN= option with SAS/ACCESS® 9.2 engines. *SAMPLES & SAS NOTES.* URL: http://support.sas.com/kb/38/204.html

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