Secure Macro-Based Method to Assign LIBNAMEs for Databases

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

ABSTRACT

The use of meta-LIBNAMEs in the client-server environment is one way to securely assign LIBNAMES, i.e., the user does not know (and cannot acquire) the associated userids or passwords. Sherman & Carpenter (2009) present a 3-step SQL-based approach that does not require metadata. Their approach works with PROC SQL CONNECT TO syntax for data pulls, but does not work for assigning LIBNAMEs to relational databases. Here we describe a compiled, encrypted macro that is an enhancement and extension of their work; an approach to securely assign LIBNAMEs for relational databases. This approach uses an encrypted SAS® file that contains the login parameters; the password or encryption keys and the location (path/name) of the parameter file are hidden from the user. Compiled, encrypted macros are not as secure as one might expect and methods are described to check for and mask sensitive parameters in the executable form of compiled macros. The LIBNAME function is used in DATA _NULL_ steps, and problems in its use are described, along with the solutions. Suppressing error messages is difficult so the macro has extensive error checking logic. The macro also clears the intermediate variables used to protect against post-run examination of memory contents. Finally we close with a discussion of implementation steps and suggestions to consider to enhance security.

FUNCTIONAL REQUIREMENTS

The primary objective is to develop a method in Base SAS® to securely and silently assign LIBNAMEs/librefs to relational database systems (RDBMS). Secure & silent means the programmer can assign the LIBNAME and does not know and cannot acquire, sensitive database connection parameters (userids and passwords). The method should **not** protect database path, DSN (for ODBC), or schema names as programmers need this information.

If database connection parameters are saved in files, those files must be encrypted and the location of those files must not be revealed in the log. Both SAS-proprietary and AES encryption should be supported for parameter files.

The method should not **require** metadata but should be able to interact with metadata if needed, i.e., it should support the ability to replace user-defined LIBNAMEs with autoexec or user-defined meta-LIBNAMEs (when appropriate). The method should be able to support multiple database parameter files, e.g., a main or central file controlled by admins, and user-controlled files.

To the maximum extent possible, the method should be secure:

- If implemented via a macro, then SAS options that disclose the inner workings of macros (e.g., MPRINT) should not reveal any sensitive information,
- known hacks like the following should not disclose sensitive information:
  - %PUT %macro_name_here;
  - %PUT _ALL_; and
  - LIBNAME _ALL_ LIST;

- the method should work with SAS proprietary encryption and also with AES encryption (SAS 9.4+; not available in earlier releases).

To clarify security, the following are to be protected and kept secret from users:

- Userids for target databases,
- Passwords for target databases,
- Location/name of (encrypted) database parameter files,
- Passwords or encryption keys for encrypted database parameter files.

Source code for the method is not secret, so long as the above items are protected.

BACKGROUND

Metadata-based approach. The easiest way for programmers to access database systems without knowing the userids and passwords is – in a client-server, metadata environment – to have your admins:

- Register the database sources in metadata using SAS Management Console (note: if you are using SAS Data Integration Studio, you may be able to register databases with that product)
- Define meta-LIBNAMEs to the target databases;
- Include the meta-LIBNAME statement in the relevant autoexec files.

SAS Management Console can also be used to assign and manage permissions for the meta-LIBNAMES. Programmers working in the client-server environment can then access the databases by using the librefs associated with the meta-libnames.

The pros and cons of the metadata approach are straightforward:

Pro:

- Burden of creating and maintaining the LIBNAMEs is shifted to admins and/or programmers with sufficient privileges in SAS Data Integration Studio
- Easiest approach for programmers who don't use/lack privileges in SAS Data Integration Studio

Cons:

- May be slightly less secure as admins are usually a central function and security is tighter with information compartmentalization
- Constrained to the client-server environment, i.e., not portable to or useful for PC SAS users and many mainframe users
- Might not work for SAS jobs running in batch on servers that support a metadata environment, as metadata might not be setup for batch.

Sherman & Carpenter (2009) method. The paper by Sherman & Carpenter (2009) presents a secure method to access databases that does not require metadata and works in any environment. It is a 3-step SQL-based approach; in a single PROC SQL invocation, their method is to:

1. Pull database login parameters from a SAS file (unencrypted in early versions of their paper), and save in macro variables
2. Use the database connect parameters with PROC SQL CONNECT TO syntax, to perform the target data pull
3. Follow-up with another SQL SELECT statement to clean up (erase) the relevant macro parameters.

They suggest using step 2 to issue a LIBNAME function call via an SQL SELECT statement. Unfortunately, their method does not work with LIBNAMEs for relational databases: the LIBNAME

function cannot have null arguments in PROC SQL and one argument to the function call must be null for databases (i.e., the physical directory associated with the LIBNAME). This contradiction prevents use of their excellent method for the task of assigning LIBNAMEs to relational databases. (Their method will work for LIBNAMEs for SAS files, but simpler methods are available and more relevant for that application.)

Their paper/method, along with the resultant challenge to securely assign LIBNAMEs to RDBMS systems, inspired and motived the development of the methods in this paper. Additionally, our goal was to protect database login parameters in a secure, encrypted SAS file. This created additional challenges, i.e., the need to make sure the encryption keys/passwords for encrypted SAS files are kept secret.

ADDITIONAL SECURITY AND OPERATIONAL FEATURES

We modified the approach of Sherman & Carpenter (2009), replacing some of their SQL with DATA _NULL_ steps, and also used encrypted database parameter files for greater security. The code for the method is encapsulated in a macro. Our approach induces additional operational design features and requirements.

Compiled, encrypted SAS macros are not secure. The use of a compiled, encrypted macro does **not** guarantee security for the source code. Opening a compiled macro file in a text editor like notepad or vi reveals that some/most of the SAS macro statements are masked in the compiled file, but SAS non-macro code may appear as plain text on the editor screen. Some macro statements may be visible as well.

This is not surprising – if you similarly look at compiled program files in other computer languages, character constants often appear as plain text constants in executable files. From the perspective of a SAS macro, the non-macro SAS code is (presumably) plain text constants to be passed-through, as-is.

For much of our macro code, this is not an issue. However it is an issue for sensitive information like userids, passwords, and encryption keys. Experimentation led to the following solution when only a single macro is being compiled. Sensitive parameters, say e.g., passwords/encryption keys in code like this:

```
SET mylib.myfile (READ=secret);
SET mylib.myfile (ENCRYPTKEY=supersecret);
```

may be clearly visible as plain text in the compiled, encrypted macro.

To protect the information, use the following approach: define %LOCAL variables for the sensitive parameters and use those macro variables in the code, taking the place of the sensitive information text:

```
%LOCAL my_password my_key;
%LET my_password=secret;
%LET my_key=supersecret;

SET mylib.myfile (READ=&my_password.);
SET mylib.myfile (ENCRYPTKEY=&my_key.);
```

Looking at the code above in a compiled macro in a text editor may show the strings &my_password and &my_key, but not the actual (confidential) values stored in those macro variables.

Additional experimentation showed that when compiling >1 macro at a time, the code for the last macro may be visible in the compiled file. To counteract this, the last macro compiled can be a stub macro. Exposing the code for a stub macro does not reveal any sensitive information. (A stub macro is needed for the main macro processing to work; details later herein.)

Constraints on SAS-proprietary passwords and AES encryption keys in SAS code. The code used to access an encrypted file must include:

```
READ=password or READ=&my_password
ENCRYPTKEY=key or ENCRYPTKEY=&my_key
```

as a data set options. Sherman & Carpenter use SYMGET to pull in database passwords; unfortunately this does not work for READ= or ENCRYPTKEY= access to SAS files. Similarly in this context, the RESOLVE function does not work either. That is, this code will **not** work and will cause an error message:

```
READ=SYMGET('password')
ENCRYPTKEY=RESOLVE('&my_key').
```

SAS proprietary passwords are limited to valid SAS names, 8 characters or less, and are not case sensitive; this is generally considered to be a weak password. SAS provides 3 ways to supply AES encryption keys, which can be up to 64 characters in length:

1. `ENCRYPTKEY=string or ENCRYPTKEY=&my_string`
2. `ENCRYPTKEY="string" or ENCRYPTKEY="&my_string"`
3. `ENCRYPTKEY='string'.`

Options 2 & 3 allow blanks and special characters which can complicate their use in macros (especially trailing blanks). Experimentation led us to select option 1 with no special characters; it is the weakest key of the 3, but it is still highly secure, avoids the problems of the other options, and works best in operation.

PROC PWENCODE can be used to encode passwords for relational databases, but SAS-proprietary data set passwords that are encoded this way do **not** work with the READ= (or PW=) data set options, and also do not work for AES encrypted files and the ENCRYPTKEY= data set option.

Passwords and encryption keys are not masked in macros and the methods described above should be used to protect those parameters in compiled, encrypted macros.

Notes can be suppressed but warnings and error messages are difficult to suppress. Notes and the display of source code in the log can be suppressed, but it is more difficult to suppress error messages and warnings. Because of this, the macro has extensive error checking and handling, to prevent disclosure of sensitive information when errors occur.

As a general principle, the level and degree of error checking done should reflect the importance of the program and the underlying requirements. The sensitive parameters here are slowly changing dimensions that are under control of admins and/or programmers. Admins should monitor changes in database login credentials; a test program that checks every database connection, 1X every day, is recommended for operations.

Database parameter files: support multiple databases and environments. Database parameter files are used to contain the database login parameters. Each database & environment is a separate row in the file. Two variables uniquely identify each row/connection:

- source_name = label for or name of database
- env = intended to be the run environment, i.e., dev, test, prod, but can be anything you want.

The database parameter file also contains in each row, variables that specify:

- A default libref to be assigned (used when no libref is supplied by the user or an invalid libref is specified),
- Userid and password for the remote database
- SAS data engine to use – Oracle, ODBC, etc.

- Schema name for the remote database
- Path (set to missing for ODBC connections)
- DSN (ODBC only; set to missing for non-ODBC)
- An override indicator, described below.

Variable lengths as used in macro development are:

```
length source_name env $30.;
length default_libn $8. db_user engine $30.;
length pass $300.;
length path schema dsn $40. override $5.;
```

Libref override. Two operational situations need to be accommodated:

- Programmers use their own librefs during development (and they should be macro variables for portability) but a uniform libref is to be used when the programs go into production, and/or
- Over time, a data source may be assigned a meta-LIBNAME and programs should use the meta-LIBNAME.

The above is supported as follows:

- Each row in the database parameter file has a variable for a default libref and an override indicator. When the override indicator is set (to yes), the user-supplied libref is ignored and the default libref is used for the macro processing. If no libref is supplied by the user, the default value in the parameter file is used.
- The macro returns the assigned libref in a global macro variable, and any associated error indicators
- For the above to work optimally in practice, users should write their own macro to invoke the main macro and check the resultant error indicators and use the assigned libref as returned OR abort the run if appropriate. This is not absolutely necessary but is recommended.
- The above will encourage those who use hard-coded, non-macro-variable librefs to change their code.
- In the database parameter file, if the variable override is set to ‘no’ and the user supplies a valid libref, that libref will be used by the macro.

Need to create encrypted parameter file. The use of an encrypted database parameter file means that:

- The admin and/or users who wish to have their own parameter file will need to create an encrypted file that provides the relevant database connection parameters. To maintain security, database connection parameters should be tested before (and after) the information is added to the parameter file.
- The LOGs for programs that create or update database parameter files should be deleted, for the obvious security reasons.
- For AES encryption, a program that creates a random encryption key (user-selected length; 64 characters is recommended) was developed. Use of this program is optional; see Appendix 2 for the program code.

Admin-owned & user-owned parameter files. The requirement to support multiple database parameter files (admin-owned and/or user-owned) is supported as follows.

- The location/name of the main or default database parameter file is specified in the main macro.
- Users who wish to provide/use their own database parameter file will need to write a macro that provides the location/name of their file. The macro must have a pre-set name: %PA_LIBR, and it too must be complied and encrypted – but in a separate, user-supplied library (code for the macro is below).

- A “stub” version of the %PA_LIBR macro is stored in the compiled macro library that contains the main macro. This stub version returns nothing, and is used as a signal in controlling when the default parameter file location is used.
- Library/catalog concatenation is used to cause the user-defined version of PA_LIBR to be invoked instead of the stub version:
  - The user must specify libnames pointing to both macro libraries, #1 that contains their version of %PA_LIBR and #2 the main library with the stub version of the macro,
  - then specify a 3rd libname or catname for the concatenated libraries (#1 and #2).
  - The concatenated libref or catref is then specified with OPTIONS MSTORED SASMSTORE=libref or catref.
  - If the user-defined libref is the 1st library in the concatenation, then the user-defined version of %PA_LIBR will be invoked/used instead of the stub/default version

Sample SAS code to concatenate compiled macro libraries: in the code below, the user library is listed first in the concatenation, so a user-defined version of %PA_LIBR will be invoked instead of the stub version.

```
libname stdmcr "path_to_main_library"  access=readonly;
libname stdmcr2 "path_to_user_library" access=readonly;
libname mcrolib (stdmcr2 stdmcr) access=readonly;
options mstored sasmstore=mcrolib;
```

Users who do not need to provide their own database parameter file can ignore the above and instead rely on the default database parameter file.

SAMPLE MACRO TO SUPPLY LOCATION INFORMATION FOR USER-OWNED DATABASE PARAMETER FILE

The code is below; fill in the location information as needed (i.e., macro variables at top of program). Notice the test of &sysuserid against the %local macro variable value. As written, this macro (deliberately) works for 1 and only 1 userid. Of course it can be modified to support multiple userids if needed; the SAS macro language supports an IN operator if the MINOPERATOR system option is selected (also see related option MINDELIMITER=).

Comments in the macro code provide an explanation of variables. The multiple %let statements contain the sensitive parameters. If a password or any of the parameters below contains a special character that can throw errors in the macro language, you will need to modify your code to use the appropriate macro quoting functions. Also note use of %superq function - that might not be required in some cases, but is suggested to reduce the risk of run-time errors in macro execution. The macro variables are cleared on exit, to prevent disclosure by looking in memory after execution.

Note: The SAS code below is released under a *BSD 2-clause open source copyright license*; see Appendix 1 for the license text.

```
%macro pa_libr(nbr) / store secure;
      %local usr pth fnm psw ekey;
      %let usr=*****;    * fill in:  authorized userid;
      %let pth=*****;    * fill in:  path to file;
      %let fnm==*****;   * fill in:  SAS data set name;
      %let psw==*****;   * fill in:  SAS-proprietary password;
      %let ekey=*****;   * fill in:  AES encryption key for file;

      %if %length(&nbr.) = 0 %then
            %goto exit;
```

```
      %if ((&nbr. ne 1) and (&nbr. ne 2) and (&nbr. ne 3) and (&nbr. ne 4))
%then
            %goto exit;

      %if (&sysuserid. = &usr.) %then
            %do;
                  %if (&nbr. = 1) %then
                        %superq(pth);

                  %if (&nbr. = 2) %then
                        &fnm.;

                  %if (&nbr. = 3) %then
                        %superq(psw);

                  %if (&nbr. = 4) %then
                        %superq(ekey);
            %end;

%exit:
      %let usr=;
      %let pth=;
      %let fnm=;
      %let psw=;
      %let ekey=;
%mend;
```

The stub version of the macro is as follows:

```
%macro pa_libr(nbr) / store secure;
%*macro pa_libr(nbr);
      %* stub macro, returns nothing;
      %return;
%mend;
```

MAIN MACRO TO ASSIGN LIBNAMES TO RDBMS SYSTEMS (ANNOTATED)

The source code for the main macro follows. Additional comments on the processing are highlighted. Sensitive information (userids, passwords, etc.) are redacted and replaced with other text in the material below.

System options. For write access, use: `libname stdmcr "/path_goes_here/";`
For ALL other macro uses, **access to the compiled macro library should be read-only**:
`libname stdmcr "/path_goes_here/" access=readonly;`
These system options are required to support compiled macros: `mstored sasmstore=`

```
libname stdmcr "/path_goes_here/"; *<- location of compiled macro library;
options nocenter mstored sasmstore=stdmcr dtreset;
```

Macro header. Also capture values of relevant system options and save in macro variables, then turn off all system options that reveal information about the workings of the macro. MACROGEN is not turned off; it is an obsolete option that does not throw an error. Specifying MACROGEN does **not** reveal any information when MPRINT, MLOGIC, SYMBOLGEN are turned off. (This has been tested/confirmed in 9.4.)

```
%macro rdb_libname(source_name=, env=, mylib=, enctype=aes, pmfile=default) /
store secure;
%*macro rdb_libname(source_name=, env=, mylib=, enctype=aes, pmfile=default);
      %*;
      %* SAS macro to assign/link databases to a SAS program, in a secure
manner (details below).;
      %* Database access parameters are stored in an encrypted, password-
protected;
      %* SAS file.  The basic steps here are:;
      %*   1. Generate random libref to use to link to parameter file.;
      %*   2. Use LIBNAME function to setup access to parameter file.;
      %*   3. Use SQL to pull record for target database from the parameter
file,;
      %*       save access parameters in (local) macro variables. Deassign
LIBNAME after use;
      %*   4. Use macro variables from 3 to issue LIBNAME function to provide
a link to;
      %*      the target database.;
      %*   5. Clear memory for extra security. Report results to user via log
and
      %*      specific global macro variables.;
      %*;
      %*  Version 1.0, Thomas Billings, June 2016;
      %*;
      %*  Secure manner = want to keep these things secret:;
      %*                  name + location of database access parameter file;
      %*                  userid + password used to access target database.;
      %*  Want to keep these out of the log, minimize them in source (which
will be compiled;
      %*  and encrypted), and not accessible by system options.;
      %*;
      %*  Macro inputs, keyword parameters:;
      %*   source_name = name or handle for target database (required);
      %*   env = environment, usually one of:  prod, dev, test, audit, etc.
Not required;
      %*         defaults to prod;
      %*   mylib = libref you want to use.  Not required, can be overridden
by setting;
      %*           override=yes, in which case the default libref in the
parameter file;
      %*           will be used;
      %*   enctype = sas or aes.  encryption type of database parameter
file.;
      %*             aes encryption is default, available only in 9.4+, not
valid in 9.3;
      %*   pmfile = default or user.  see comments above on macro plus file;
      %*;
      %*  Macro outputs, global macro variables:;
      %*   rdb_libname_rc = return code from the macro. Any value other than
0 is an anomaly.;
      %*   rdb_libname_msg = return message from the macro, identifies
anomalies.;
      %*   rdb_libref = libref used by the macro. Not necessarily the same as
user-requested input.;
      %*;
      %*  Suggested macro usage:;
```

```
      %*  %rdb_libname statement for target database, followed by a %let
statement;
      %*  that assigns the value of &rdb_libref, to your macro variable for
the target;
      %*  database libref;
      %*;
      %* global macro variables for libref, return code and message;
      %global rdb_libname_rc rdb_libname_msg rdb_libref;
      %global SYSDBMSG;

      %* macro variables for user-specified options;
      %local mprint mlogic symbolgen source source2 notes;

      %* capture/save values of user-specified options;
      %let mprint=%qsysfunc(getoption(mprint));
      %let mlogic=%qsysfunc(getoption(mlogic));
      %let symbolgen=%qsysfunc(getoption(symbolgen));
      %let source=%qsysfunc(getoption(source));
      %let source2=%qsysfunc(getoption(source2));
      %let notes=%qsysfunc(getoption(notes));

      %*  turn off source/log/macro print options;
      options nomprint nomlogic nosymbolgen nosource nosource2 nonotes;
```

Macro housekeeping and **get location of database parameter file –> user-supplied or default**.

```
      %*  check parameter pmfile and adjust;
      %let pmfile = %upcase(%qtrim(%qleft(&pmfile.)));

      %if (%length(&pmfile.) = 0) %then
            %let pmfile=DEFAULT;

      %* variables re:  database access parameters;
      %*  also: password/encryption key for database parameter access file;
      %local pa_dir pa_sds pa_psw pa_dsn pa_ekey pa_lib;
      %local ref_pfx_char refck pflibrc pflibnm mylibref pswd_ekey;
      %local pm_ck;

      %let pa_dir=%pa_libr(1);

      %if ((&pmfile. ne DEFAULT) and (%length(&pa_dir.) ne 0)) %then
            %do;
                  %let pa_dir=%pa_libr(1);
                  %let pa_sds=%pa_libr(2);
                  %let pa_psw=%pa_libr(3);
                  %let pa_ekey=%pa_libr(4);
                  %put USER_SUPPLIED PARAM FILE;
            %end;
      %*********************************************************;
      %***** ADMIN TO SET DEFAULT VALUES FOR PARAMETERS BELOW;
      %else
            %do;
                  %let pa_dir=fill_in_path_here;
                  %let pa_sds= fill_in_parameter_data_set_name_here;
                  %let pa_psw= fill_in_SAS_proprietary_password_here;
                  %let pa_ekey= fill_in_AES_encryption_key_here;
                  %PUT DEFAULT PARAM FILE;
```

```
            %end;
```

More macro housekeeping plus **default processing for macro keyword parameters**.

```
      %* prefix character for libref to database access file;
      %let ref_pfx_char=M;
      %*******************************************************;
      %let usz=;
      %local dlibn eng usz psw sch ovr dsn;

      %* set global macro return code. 0=no error.;
      %let rdb_libname_rc=99;
      %let rdb_libname_msg=;
      %let rdb_libref=NOT_ASSIGNED;

      %* check user-supplied parameters;
      %let source_name = %upcase(%qtrim(%qleft(&source_name.)));
      %let env = %upcase(%qtrim(%qleft(&env.)));
      %let mylib = %qtrim(%qleft(&mylib.));
      %let enctype = %upcase(%qtrim(%qleft(&enctype.)));

      %if (%length(&env.) eq 0) %then
            %let env=PROD;

      %if (%length(&enctype.) eq 0) %then
            %let enctype=AES;
      %let pm_ck = 0;
```

If user-supplied libref, check for validity using NVALID function inside a DATA step (this function should not be used with %sysfunc or %qsysfunc). Note use of NOLIST DATA step option to suppress printout of rows if errors occur. Can use system option ERRORS= to do same but that is more work.

```
      %* if mylib= pararameter supplied, check for validity;
      %if (%length(&mylib.) ne 0) %then
            %do;

                  data _null_ / nolist;
                        x = nvalid(strip(resolve('&mylib')),'V7');
                        call symputx('pm_ck',x);
                        call missing(x);
                  run;

                  %if (&pm_ck. = 0) or (%length(&mylib.) > 8) %then
                        %do;
                              %let rdb_libname_rc=-1;
                              %let rdb_libname_msg=Error in macro inputs,
invalid mylib parm.;
                              %goto mc_exit;
                        %end;
            %end;
```

Even more housekeeping, plus **generate random libref to use to point to the database parameter file**. Note use of RAND function instead of RANUNI; ref. Wicklin (2013). The random number is rescaled to avoid overflow of digits; the end result is an 8-character libref: a letter (set via a %LOCAL macro variable)

followed by a random 7-digit number. Early versions of this macro asked the user to supply a libref; the approach here was adopted to avoid potential collisions with existing, in-use librefs.

```
      %* trim off any extraneous blanks;
      %let pa_dir = %qtrim(%qleft(&pa_dir.));
      %let pa_sds = %qtrim(%qleft(&pa_sds.));
      %let pa_psw = %qtrim(%qleft(&pa_psw.));
      %let pa_ekey = %qtrim(%qleft(&pa_ekey.));

      %*  generate random libref for database access parameter file;
      data _null_ / nolist;
            length crand7 $7. ref $8.;

            * up to 7-digit random number;
            rand7 = round((rand('UNIFORM') * 0.99999e7),1);
            crand7 = put(rand7,z7.);
            ref = cats(strip(resolve('&ref_pfx_char')),crand7);
            call symputx("pa_lib",ref);
            call missing(rand7,crand7,ref);
            stop;
      run;
```

Define LIBNAME that points to the database parameter file; check that target SAS file is present in directory. Error handling for exceptions.

```
      %* define LIBNAME to point to database parameter file;
      data _null_ / nolist;
            x =
libname(strip(symget('pa_lib')),strip(symget('pa_dir')),,"access=readonly");
            call symputx('pflibrc',x);
            call missing(x);
      run;

      %* error handling - if LIBNAME failed;
      %if (&pflibrc ne 0) %then
            %do;
                  %let rdb_libname_rc=-3;
                  %let rdb_libname_msg=Error in accessing database parameter
file.;
                  %goto cl_exit;
            %end;

      %* now check to make sure SAS data set is present in LIBNAME/directory;
      %let pflibnm=&pa_lib..&pa_sds.;

      %if (not %qsysfunc(exist(&pflibnm.))) %then
            %do;
                  %let rdb_libname_rc=-4;
                  %let rdb_libname_msg=Error in accessing database parameter
file.;
                  %goto cl_exit;
            %end;
```

At this point we have a LIBNAME and a database parameter file. Now **use SQL with variables that identify the target row, to pull the desired database connection parameters** from the encrypted file. This is per Sherman & Carpenter (2009). For security, clear the libref after the pull, even if it threw an error.

```
      %if (&enctype. = AES) %then
            %let pswd_EKEY=%STR(encryptkey=)&pa_ekey.%str( );
      %else %let pswd_EKEY=%STR(read=)&pa_psw.%str( );

      %* now we have a libname and data set pointing to database parameters;
      %* Use PROC SQL to pull database access credentials from SAS data set
and store in;
      %* macro variables.;
      proc sql noprint noerrorstop nofeedback;
            select default_libn, engine, db_user, pass, dsn, path, schema,
override
                  into :dlibn, :eng, :usz, :psw, :dsn, :pth, :sch, :ovr
                        from &pflibnm. %unquote((&pswd_ekey.))
                              where (upcase(source_name) = "&source_name.")
and
                                    (upcase(env) = "&env.");
      quit;

      %* clear LIBNAME for parameter file;
      %* this is done even if sql fails, to prevent exposure of location of
parameter file;
      data _null_ / nolist;
            y = libname(strip(symget('pa_lib')));
      run;
```

SQL pull of database connection parameters may throw an error or return nothing. Check and pursue an error exit if warranted.

```
      %* check for error in sql pull of parameters;
      %if ((&sqlrc. ne 0) and (&sqlrc. ne 4)) %then
            %do;
                  %let rdb_libname_rc=-5;
                  %let rdb_libname_msg=Error in accessing database parameter
file.;
                  %goto mc_exit;
            %end;

      %* check for no data from sql pull;
      %if (%length(&usz.) = 0) %then
            %do;
                  %let rdb_libname_rc=-6;
                  %let rdb_libname_msg=Parameters for target not found in
database parameter file.;
                  %goto mc_exit;
            %end;
```

Determine proper libref name per database parameter file and user-supplied libref. **Check if libref already assigned,** terminate if necessary.

```
      %* override user-specified libref per database parameter file;
      %let mylibref=%qtrim(%qleft(&mylib.));

      %if (%length(&mylibref.) = 0) %then
            %let mylibref=&dlibn.;
```

```
      %if (%length(&ovr.) ne 0) %then
            %do;
                  %if (%upcase(%qtrim(%qleft(&ovr.))) = YES) %then
                        %let mylibref=&dlibn.;
            %end;

      %* check if designated libref already assigned = terminate without;
      %* assigning database LIBNAME;
      %if (%qsysfunc(libref(&mylibref.),1.) = 0) %then
            %do;
                  %let rdb_libname_rc = -2;
                  %let rdb_libname_msg=LIBNAME for database parameters in
use. Macro terminated.;
                  %let rdb_libref=&mylibref.;
                  %goto mc_exit;
            %end;
```

Issue LIBNAME for target database. Note construction of LIBNAME function call puts the connection parameters into a single character variable, for use in the function call. Putting the construction logic in the LIBNAME function call caused problems in test runs as it was not always parsed correctly – i.e., we had a beautiful and correct connection string that threw SAS and database errors. Putting it into an intermediate variable fixed that problem. Note that intermediate variables are cleared after use, for increased security. **Check for successful connection and set macro variables for results. The output libref is in global macro variable &rdb_libref**

```
      %* now we are ready to issue LIBNAME function for the target database;
      data _null_ / nolist;
            length lbn $8. engine $30.;
            length LBopts $600. path_dsn $100.;
            lbn = symget('mylibref');
            engine = symget('eng');
            if (not missing(symget('pth'))) then
                  path_dsn = cat('path = ','"',strip(symget('pth')));
            else path_dsn = cat('dsn =','"',strip(symget('dsn')));
            LBopts = cat('user = ','"',strip(symget('usz')),'" ','pass =
','"',strip(symget('psw')),'" ',
                  strip(path_dsn),'" ','schema =
','"',strip(symget('sch')),'"');
            x = libname(strip(lbn),,strip(engine),strip(LBopts));
            call symputx('rdb_libname_rc',x);
            call missing(lbn,engine,LBopts,path_dsn,x);
      run;

      %if (&rdb_libname_rc. = 0) %then
            %do;
                  %let rdb_libname_msg=Normal termination.;
                  %let rdb_libref=%qtrim(%qleft(&mylibref.));
            %end;
      %goto mc_exit;
```

Error exit and normal exit. Note that &syslast and &SYSDBMSG are both cleared, along with all important/sensitive %local variables.

```
%cl_exit:
```

```
      %* try to clear LIBNAME for parameter file;
      %* this is done even if attempt fails, to prevent exposure of location
of parameter file;
      data _null_;
            y = libname(strip(symget('pa_lib')));
      run;

%mc_exit:

      %* precaution:  clear sensitive data from memory;
      %let syslast=dataset_info_redacted;
      %let pflibnm=;
      %let usz=;
      %let psw=;
      %let pa_dir=;
      %let pa_lib=;
      %let pa_sds=;
      %let pa_psw=;
      %let pa_ekey=;
      %let pswd_ekey=;
      %let SYSDBMSG=Redacted by rdb_libname macro.;
```

Print information messages. Restore system options saved at beginning of macro.

```
      %* let user know if success (or not) and database assignment;
      %put Macro rdb_libname: return code = &rdb_libname_rc.;
      %put Macro rdb_libname: libref = &rdb_libref.;

      %if (&rdb_libname_rc. = 0) %then
            %do;
                  %put Macro rdb_libname: normal termination.;
                  %put Macro rdb_libname: Engine = &eng.;
                  %put Macro rdb_libname: Path (N/A for ODBC) = &pth.;
                  %put Macro rdb_libname: ODBC DSN = &dsn.;
                  %put Macro rdb_libname: Schema = &sch.;
            %end;

      %if (&rdb_libname_rc. ne 0) %then
            %put Macro rdb_libname: return message = &rdb_libname_msg.;

      %*  restore source/log/macro print options;
      options &mprint. &mlogic. &source. &source2. &symbolgen. &notes.;
%mend;
```

STEPS IN IMPLEMENTATION

The ordering below is not strict; some things can be done in parallel (only main steps are listed).

1. If not already in-hand, get access to and valid/tested login parameters for at least some of the target relational database environments of interest.
2. Create AES encryption key (see Appendix 2) or determine SAS proprietary password
3. Create an encrypted database parameter file. Set/check relevant operating system (and/or metadata) permissions on the file.
4. Migrate a copy of the macro source code to a directory/file or source code management repository, e.g., Git, that is restricted access/use.

5. Modify the macro program code to add sensitive parameters (SAS proprietary passwords, encryption keys, database parameter file path/name). Compile the macros. (Main macro and %PA_LIBR supplementary macro if needed.) Verify that sensitive data cannot be seen in the compiled file.
6. Test that the macro works -- access to target databases
7. Delete logs from the macro compilation step(s) as those contain sensitive information. If you are using a tool like SAS Enterprise Guide that saves logs in the .egp file by default, you may need to take additional steps to delete the logs.
8. Write wrapper macro to invoke the main macro to setup LIBNAMEs (optional; recommended)
9. Change relevant programs – if necessary – to invoke the wrapper macro and use the librefs setup by the secure LIBNAME macro.

Implementation has a cost in terms of setup effort, but once running, the process should require only limited maintenance as database parameters are usually (very) slowly changing dimensions.

IMPLEMENTATION ISSUES

Debug is a challenge. Because the macro is compiled and produces no log, no notes (other than macro-generated messages), the macro cannot be debugged as-is when errors occur. Instead, for debug, create a new version of the macro that:

- Is **not** a stored, compiled macro
- Change the code so MPRINT, MLOGIC, SYMBOLGEN work normally.
- Rerun the uncompiled version to try to reproduce the error message(s).

After debugging, create a revised version of the stored, encrypted, compiled macro, and delete the uncompiled version.

Encrypted files are more difficult to work with:

- To open an encrypted file in most SAS tools or code, you must enter the encryption key or password (or include it in code), depending on the type of encryption used.
- If you don't have a password or encryption key, you can't do much with encrypted files.
- Any SAS logs that show encryption keys or passwords should be deleted. Similarly, sensitive parameters should be deleted from source code files as well (can delete after runs).
- Changing a file using online SAS data editors may leave no log; consider this option if/where relevant.

The exception here is meta-LIBNAME files that are metadata-bound and encrypted. For these, encrypted file access is handled automatically by the SAS system – no need to enter passwords or keys if you have the proper metadata access. However if the database parameter file is encrypted this way, then anyone who uses the macro has read access to the database parameter file – not a good idea for security!

How to control access to the macro?

- Use your operating system permissions (for the compiled macro file and/or database parameter files)
- Use metadata permissions in a client-server environment (control access to database file)
- Modify the main macro to check &sysuserid against a list of authorized users. Constraint: this approach requires ongoing maintenance of an authorized user list, something to be avoided when possible.

How to monitor changes to the database parameter files?

- PROC DATASETS AUDIT feature can be used but as the file contents are very slowly changing dimension variables, may be more complex than is needed.
- Advanced users: consider SAS Environment Manager, SAS Logging Facility.

FUTURE CONSIDERATIONS

- In an email exchange, Art Carpenter suggested replacing some of the DATA _NULL_ steps used to invoke the LIBNAME function with macro code that uses %SYSFUNC paired with %SUPERQ. Readers may wish to experiment with this; it might yield cleaner code.
- The DATA _NULL_ steps in the macro are not compiled and hence do not use the SOURCE=NOSAVE or SOURCE=ENCRYPT options as it is not necessary for our application. Others may find these potentially useful and/or might be used to prevent disclosure of source code. (These options work only with compiled DATA steps whereas the methods described above presumably work with any SAS code in a compiled, encrypted macro.)
- The macro was developed for and tested with database applications that are read-only. Changes may be needed for applications that require write access, e.g., additional options may need to be supplied in the LIBNAME function calls.
- The macro uses the EXIST function to test for existence of the database parameter file. Hughes (2016) shows that this function can fail in multi-processor parallel environments. The database parameter file is used in read-only access in the main macro. It is updated only on occasion, by admins or users who control their own parameter files. Under these circumstances, the probability of simultaneous write access conflicts is effectively zero, hence in this context EXIST is safe to use. Note however that Hughes (2016) provides alternative methods that could be used, if appropriate.

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;

/*
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: SAMPLE CODE TO GENERATE RANDOM AES ENCRYPTION KEY

```
options nocenter;

** The macro variables are for tuning the run - set to whatever;
** is appropriate for your situation;

%global KL NR FC;
%let KL=64;      *<- length of key;
%let NR=100;     *<- number of random cases to generate per character;
%let FC=X;       *<- 1st character for string. Cannot be numeric;

data mykey (keep=RC);
      length alphabet $26.  numeric $10. RC $1.;;
      input alphabet;
      input numeric;

      do i=1 to 26;
            RC = substr(alphabet,i,1);
            output;
            RC = upcase(RC);
            output;

            if (i le 10) then
                  do;
                        RC = substr(numeric,i,1);
                        output;
                  end;
      end;

      stop;
      datalines;
abcdefghijklmnopqrstuvwxyz
0123456789
;

data rand_key;
      set mykey;

      do i=1 to &NR.;
            rn = rand('UNIFORM');
            output;
      end;

      drop i;
run;

proc sort data=rand_key;
      by rn;
run;
data keygen;
      set rand_key (obs=&KL.) end=kend;
      length aes_key $&KL..;
      retain aes_key;
      aes_key = cats(aes_key,RC);

      if (kend) then
```

```
            do;
                  substr(aes_key,1,1) = "&FC.";
                  output;
                  put @1 aes_key $&KL..;
            end;

      keep aes_key;
run;
```

REFERENCES

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

Hughes T M (2016). A Failure To EXIST: Why Testing for Data Set Existence with the EXIST Function Alone Is Inadequate for Serious Software Development in Asynchronous, Multiuser, and Parallel Processing Environments. *MWSUG 2016 Conference Proceedings.* URL: http://www.mwsug.org/proceedings/2016/BB/MWSUG-2016-BB30.pdf

Sherman P, Carpenter A (2009). Secret Sequel: Keeping Your Password Away from the LOG. *SAS Global Forum Proceedings.*
URL: http://sascommunity.org/wiki/Secret_Sequel:_Keeping_Your_Password_Away_from_the_LOG

Wicklin R (2013). Six reasons you should stop using the RANUNI function to generate random numbers. SAS Blog: *The DO Loop.* URL: http://blogs.sas.com/content/iml/2013/07/10/stop-using-ranuni.html

ACKNOWLEDGEMENTS

Thanks to Paul Sherman and Art Carpenter for their informative 2009 paper. Also, thanks to the following for valuable comments and suggestions:

- Viraj Kumbhakarna, MUFG Union Bank, N.A.
- Art Carpenter, California Occidental Consultants

Any errors herein are solely the responsibility of the author.

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