SESUG Paper 164-2020

Data Security in SAS® using Encryption and Hashing

Rajasekar Sundaram
Kiran Venna, SMACT Works, Inc

Abstract

In the current digital world, restricting unauthorized access to sensitive data is of paramount importance. SAS® provides different methods to secure data. The main objective of this paper is to describe two major methods to protect SAS® datasets from the data security breach. The first method is about providing controlled user access to the datasets through encryption. The second method is about masking confidential information with the help of the hashing technique. By combining both encryption and hashing, unauthorized data access can be restricted. This paper will describe briefly both encryption and hashing technique.

Key words

Encryption, Hashing functions, AES, Data Security

Introduction

Data Security is one of the most important aspects for any organization especially in today's world of cybercriminals. In the current industry trend often SAS® datasets might contain personally identifiable information (PII) like Social security numbers, Credit card numbers, etc. Securing PII information in SAS® datasets is very crucial for data security. There are several mechanisms to secure the data, of which encryption and hashing are important. Encryption is generally used to give restricted access of data to appropriate personnel who are authorized to process the data. Hashing is mainly involved in completely masking the data for everyone.

Encryption is one of the important techniques for data security, which usually converts human-readable text to ciphertext (unreadable text). Encryption in the SAS® can be done by various algorithms, and SAS® programmer does not need in-depth details about the algorithms. By using simple dataset options these algorithms can be implemented on a SAS® dataset. Encryption allows access of PII only to restricted users who have the required encryption key. SAS® proprietary encryption and Advanced Encryption Standard (AES) are two popular encryption techniques.

Hashing converts specific text into entirely different text. This is often defined as one-way transformation as virtually impossible to get to the original text. This is often a powerful way to secure the data as no one will be able to see the original data. MD5 and SHA256 are two popular SAS® hashing functions, which are very easy to use.

In this paper, we will discuss both encryption and hashing with the help of various examples.

Encryption:

Encrypting a SAS® dataset is an easy process, all we need to do is to mention encrypt=encryption technique and encryptkey= encryption password in SAS® dataset options. In this technique, an encryption key is required to read or edit the contents of the SAS® dataset. The encryption key can be shared with authorized users only.

In this paper, we will discuss the AES encryption technique. To apply AES encryption o nSAS® dataset we need to specify encrypt=aes in the SAS® dataset option.

Below is the code for encrypting the sample dataset. In this code, we have given 'yellow' as the encryption password.

```
data emp(encrypt=aes encryptkey=yellow);
    input name $ ssn;
    datalines;
    John  34568962
    Frank 47584264
 ;
run;
```

Once the above code is run, it generates couple of important information in the log. First one is about hiding the value of the password as shown below and the second is note about accessing the dataset in absence of encryptkey is not possible.

```
73         data emp(encrypt=aes encryptkey=XXXXXX);
74         input name $ ssn;
75         datalines;
NOTE: If you lose or forget the ENCRYPTKEY, there will be no way to open the
file and recover the data. The data set WORK.EMP has 2 observations and 2
variables.
```

If anyone tries to access the encrypted dataset without the encryption key, it will result in error. Code for accessing the encrypted dataset without encryption key is shown below.

```
data extract_emp;
      set emp;

     if name='John';
run;
```

Below is the log for the above code where encryption key is not provided.

```
data extract_emp;
  set emp;
 ERROR: Invalid ENCRYPTKEY value for WORK.EMP.DATA.

  if name='John';
  run;
```

Once user has encryption key, he can access the data by simply providing it. Below is the code for accessing the data with help of encryption key.

```
data extract_emp;
     set emp (encryptkey=yellow);

          if name='John';
run;
```

Below is the log for the above code where encryption key is provided.

```
data extract_emp;
  set emp (encryptkey=XXXXXX);

  if name='John';
run;

NOTE: There were 2 observations read from the data set WORK.EMP.
NOTE: The data set WORK.EXTRACT_EMP has 1 observations and 2 variables.
```

Even though the concept of encryption key discussed above is very effective, but it has one serious limitation. In a typical production environment, whoever has access to the code will be able to access the encryption key. Now we will discuss how to hide the encryption key from the code by storing the password in a parameter file and restricting its access.

Parameterizing Encryption password

Parameterizing the encryption password is done in 2 steps. In the first step, a macro variable is created with the value of the password, which is stored in a parameter file. This parameter file will have restricted user access along with nosymbolgen and nomprint options enabled to avoid the encryption key printed in the log. In the second step, we will use a %include statement to access the macro variable. The user who has access to this file can pass the encryption key. Below we will show the code for both of these steps.

To store a macro variable in the parameter file, first, we need to create a parameter file. Then we need to make sure this file has restricted access. For the illustration purpose in the code, we have mentioned '/home/kiranvenna1/encryptkey_for_empdataset.sas' as path for the parameter file.

In this parameter file, a macro variable is created from the encryption key by using call symput. Please note that options nomprint and nosymbolgen should be used in the parameter file so that the password is not printed in the log. Below is the code to create a macro variable from the encryption key.

```
options nomprint nosymbolgen;
data _null_;
     call symput('myencryptedpw', 'yellow');
run;
```

In the second step, we can access the stored password from the parameter file by using an %include statement and referencing macro variable myencryptedpw.

```
%include "/home/kiranvenna1/encryptkey_for_empdataset.sas";

data extract_emp;
   set emp (encryptkey=&myencryptedpw);

    if name='John';
run;
```

By using encryption and storing the encryption key in Parameter file, we can restrict unauthorized user access.

Hashing

Hashing is critical when we have confidential data in SAS® dataset. By doing hashing we can completely mask the confidential data for everyone. Hashing is generally done on few variables instead of all the variables in a SAS® dataset. Variables which contain personal information like Social security number or credit card information are ideal candidates for masking with help of hashing. Hashing masks variables by converting original text/number to entirely different text.

Now we will discuss the concept of hashing with the help of a sample dataset. In this dataset, we have 2 variables name and credit. Variable credit indicates a credit card number and has to be masked with the help of hashing. Below is the way to create a sample.

```
data test;
input name $ credit:$20.;
datalines;
Smith 1234-5670-8900
Sam 2343-5690-7890
;
```

```
run;
```

There are many hashing functions available in SAS® like MD5 SHA1, SHA256, SHA384, SHA512, and CRC32. We will be illustrating hashing functionality using SHA256. In this example, the hashing function is applied to the credit variable, and the code for the same is shown below. The hashing function does not have its format and to view the masked value we need to use $hex64. format.

```
data test;
   set test;
   credit=sha256(credit);
   format credit $hex64.;
run;
```

With the help of proc print, we can see the output and we can understand how masking helps to secure the data.

```
proc print data=test;
run;
```

Obs	name	credit
1	Smith	B156D12200DD07006A756592E5D7B61EF9F747E8
2	Sam	A446B02A62F56232BB9DFA12961EE9DFDC057BA4

From the above results, it is clear that variable credit value has been masked by converting the original text to entirely different text.

Conclusion

Encryption and hashing are critical for data security for Organizations. Encryption provides data security by restricting unauthorized user access. Proper care should be taken to successfully protect the encryption key by using a parameter file. Hashing can be leveraged to mask personal information from all the users.

References

1. SAS® 9.4 Product Documentation. http://support.sas.com/documentation/94

Acknowledgements

I would like to specially thank, Shreesh Kesharwani for giving valuable suggestions.

I would also like to thank Jatin Madaan, Dr. Mayur Savsani,Sandeep Kumar Yadav, Prem Kumar chiluveru, Kapil Mathur, Swapnil Hedau for helping us with proof reading.

Contact Information

Your comments and questions are valued and encouraged. Please feel free to contact the authors for more information.

Rajasekar Sundaram: rajasekar11285@gmail.com
Kiran Venna: kiranvenna@gmail.com