SAS-STE

(**S**ecure **T**ext **E**ncryption)

Official Documentation

SAS

Contents

About SAS-STE

SAS-STE : SAS-Secure Text Encryption is an open source text based encryption algorithm designed and made by Saaiq Abdulla Saeed (SAS).

Main features and Functions of SAS-STE

Plain text and Text based file Encryption

SAS-STE allows the encryption of pure plain text inputted to the program as well as the encryption of any text based files such as .txt files, .java files, .py files , .html files ,…etc.

Reusable keys

In addition to randomly generating public keys for encryptions SAS-STE supports the use of existing public keys (keys you have generated before for an encryption) to encrypt another file or text. This means you can use the same key to encrypt many different text-based data.

Supports Keyless Encryption and Keyed Encryptions

SAS-STE can generate new public keys or use existing public keys for an encryption, which is the keyed encryptions SAS-STE offers. However, SAS-STE also supports keyless encryptions, which is the encryption of data without any public keys. In this case the only protection you will get to your encrypted data will be from the static key embedded in the source code.

Multiple Encryptions

SAS-STE allows to automatically re-encrypt the encrypted output of the actual data based on the user's needs. This works by feeding the output of the program as a new input to the program.

Keys in SAS-STE

SAS-STE has two types of keys. Static keys and Public keys. Statics keys are embedded in the source code and can only be changed when compiling the source code. Public key is the main key you will be using while using the program. Both these keys will directly affect the encryption and decryption algorithm of SAS-STE.

Static Keys

As mentioned before Static keys are keys embedded in the source code of SAS-STE. This key is simply an integer array which will directly affect the encryption and decryption algorithm. There are a total of 4 Static keys throughout SAS-STE algorithm. You can only change these keys in the source code when you are compiling the source code yourself.

The main purpose of embedding Static keys is to make the SAS-STE copies of different users different. For instance, imaging that there are two users, USER1 and USER2. These two users have a copy of SAS-STE but with their own Static keys. So USER1 encrypts a data with his copy of SAS-STE with a randomly generated public key. Unfortunately for USER1, his encrypted data and its public key was found by USER2, and USER2 decides to

decrypt the data. However, USER2 cannot decrypt the data with his copy of SAS-STE. Why? Because USER2's SAS-STE copy doesn't have the same Static keys as USER1's SAS-STE copy.

So, in point of security even if a person manages to find a public key for an encrypted data. He cannot decrypt it unless he finds all of the Static keys or get a copy of the SAS-STE used by the person who encrypted it.

Public Keys

Public keys are the keys that will normally be used through out the usage of SAS-STE. These keys can be randomly generated for each encryption you do, or they can also be reused. Public keys of SAS-STE is a 112 character long string which stores unique information required to decrypt or encrypt data.

skey Files

'skey' files with the file extension '.skey' are the files which stores the public key of your newly generated key if you choose to save the public key to a file. You can directly give the path to a skey file when asked to input a key via file and SAS-STE will detect and take the public key from the file. In SAS-STE you can also output the newly generated key as plain text, which will make is easier to send the key to another person via SMS or email or etc. But if you are plaining on reusing a key for a specific group of data (e.g.: all source code files related to a specific project) then its is recommended to output the key as a skey file.

Changing Static keys

This part is only for developers and not for users without programing experience.

1. In order to change the static key you should first download the source code of a release of SAS-STE either from the saaiqSAS official website (https://saaiqsas.github.io) or the GitHub page of SAS-STE (https://github.com/saaiqSAS/SAS-STE).
2. Then you can open the source code via any IDE you like. And open the files layer1.java, layer2p1.java, layer2p2.java and layer2p3.java.
3. Then locate StaticKeyL1 , StaticKeyL2P1, StaticKeyL2P2 and StaticKeyL2P3 integer arrays (int[]) at the beginning of the files you opened.
4. Now download the SAS-STE-StaticKeyGen either from the SAS-STE GitHub page under related tools or in the saaiqSAS official website's SAS-STE page and extract and execute the .bat file for windows or the shell file for Linux.
5. Now copy the generated Static key and change it with the key in one of the files you opened.
6. Close the SAS-STE-StaticKeyGen and re-open/execute it to generate a new key and repeat step 5 and 6 until the static keys in the 4 files are all changed.
7. Now you can compile the source code by either running it in a java IDE or manually.
8. Now copy the output .class file and replace the .class files in the programFiles in a release of SAS-STE. Now you are good to go.

Supported Characters

Currently SAS-STE supports 95 (including SPACE) ASCII characters. Any unsupported character will not be encrypted or decrypted. Instead, it will stay as they are. But it is not recommended to encrypt a piece of data with many unsupported characters. Below is a list of all the currently supported characters by SAS-STE:

0.	13. -	26. :	39. G	52. T	65. a	78. n	91. {
1. !	14. .	27. ;	40. H	53. U	66. b	79. o	92. |
2. "	15. /	28. <	41. I	54. V	67. c	80. p	93. }
2. #	16. 0	39. =	42. J	55. W	68. d	81. q	94. ~
4. $	17. 1	30. >	43. K	56. X	69. e	82. r	
5. %	18. 2	31. ?	44. L	57. Y	70. f	83. s	
6. &	19. 3	32. @	45. M	58. Z	71. g	84. t	
7. '	20. 4	33. A	46. N	59. [	72. h	85. u	
8. (	21. 5	34. B	47. O	60. \	73. i	86. v	
9.)	22. 6	35. C	48. P	61.]	74. j	87. w	
10. *	23. 7	36. D	49. Q	62. ^	75. k	88. x	
11. +	24. 8	37. E	50. R	63. _	76. l	89. y	
12. ,	25. 9	38. F	51. S	64.`	77. m	90. z	

How SAS-STE Algorithm Work

SAS-STE algorithm has three main layers and two types of keys that directly affect the algorithm. Static keys and Public keys are the two types of keys in SAS-STE. Static keys are keys embedded in the source code and Public keys are the normally used keys through out the program. The three main layers in SAS-STE are layer0, layer1 and layer2 where layer2 has three parts layer2p1, layer2p2 and layer2p3.

What SAS-STE mainly does Is change the characters of the input into random characters. Which character should be changed to which character is decided by the keys. In addition to this SAS-STE has different security measures such as repeating (feeding output as a new input) per layer, repeating whole process, different random salting methods at different layers, increasing the original data size when encrypting, ability to use a wrong key to decrypt an encrypted data (this makes the program process normally without any errors, same as giving the actual key, but output will not be readable. This makes it harder to decrypt data without actual keys) .

Effects of Static Keys and Public keys

Public keys contain a character set generated in random order. This character set is the main set of characters that will be used to compare with the characters in the input data. As mentioned before in the Keys in SAS-STE segment, Static keys are integer arrays. This

integer array decides which place value of the character set should be changed to which place value of the character set.

For example an input data may have the character ‘A’. Then the algorithm sees which place value of the character set has the character ‘A’ and notes the place value PLACE_VALUE1. Then the algorithm sees what integer is in the place value PLACE_VALUE1 of the Static key array. Then it takes note of this integer INT1. Then the algorithm sees what character is in the place value INT1 of the character set and notes the character CHAR1. Then it substitutes the original input character with the character CHAR1.

So, as you can understand, if the Static key is changed then the integer in PLACE_VALUE1 of the Static key would be a different integer. And if the Public key is changed the place value PLACE_VALUE1 of ‘A’ would be different.

This process takes place on layer1 and the three parts of layer2 where each layer has a unique Static key.

Encryption Layers

Layer0:

Layer0 is the main layer responsible for salting. In this layer the size of the original data will be increased by adding random characters in random places. Where the characters should be added will be decided randomly and will be stored in the public key.

Layer1:

Layer1 will process the whole data under a single static key.

Layer2:

Layer2 will split the data into three parts DATA1, DATA2, DATA3 and will randomly feed these data into different parts of layer2 where each part has a unique static key.

For example, DATA1 may be fed into layer2p3 one time but the next time DATA1 may be fed into layer2p2.

Once all the data is processed by the parts of layer2, then the different data parts will be compiles into one data piece. This single compiled data piece will have encrypted output of three different static keys.

Layer2p1:

Processes the data part it receives with layer2p1’s unique Static key

Layer2p2:

Processes the data part it receives with layer2p2’s unique Static key

Layer2p3:

Processes the data part it receives with layer2p3’s unique Static key

About SAS-STE-Wizard

SAS-STE-Wizard is a step-by-step Command Line Interface (CLI) wizard which uses the SAS-STE algorithm. It is one of the interfaces to use the SAS-STE.

Details of Options in SAS-STE-Wizard

Input Type

Input type means how you want to input the data you want to process. There two methods you can input your data. As pure plain text or as a text-based file. When giving the path to a input file, make sure to give the path starting from the very root directory and to give the correct file extension (Windows: D:\\folder\\inputFile.java) (Linux: /home/user/file/inputFile.txt).

Text Input Amount

Text Input amount means how many time do you want to input pure plain text to be processed. You can choose from One time or Multiple times. In one time you get to input text once and after it is processed the program will end. However in multiple times you can get to input text many times and each time you input pure plain text it would be processed and outputted to the output type you chose. And after being outputted you will again get to input new text. If you want to end the program in multiple text input then you have to type ' !END! 'command as a text input

Function Type

Function type means what you want to do with the data you inputted. You can choose between Encrypt the data or Decrypt the data.

Key input Type

Key input type means how you want to input the key into the program. You can input it as pure plain text or as a skey file. When giving the path to a skey file, make sure to give the path starting from the very root directory (Windows: D:\\folder\\key.skey) (Linux: /home/user/file/key.skey). This option will only be available if you choose decrypt as a function and choose keyed or if you choose encrypt as function and choose use existing key.

Key output Type

Key output type means how you want to output the key generated. You can either output it as pure plain text or as a skey file When giving the path to a key output file, make sure to give the path starting from the very root directory and not to give a file extension (Windows: D:\\folder\\key) (Linux: /home/user/file/key). This option is only available if you choose to generate new key after taking encrypt as your function.

Output Type

Output type means how you want to output the final result after being processed (encrypted/decrypted). When giving the path to a output file, make sure to give the path starting from the very root directory and to give the extension of the original file for less confusion (Windows: D:\\folder\\outputFile.java) (Linux: /home/user/file/outputFile.txt)You can choose to the output as pure plain text or you can choose to save to a file.

Things to keep in mind

1. When typing ‘ \ ’ (backward slash) anywhere in the SAS-STE-Wizard make sure to type it twice ‘ \\ ’ as a single backward slash cannot be recognized by the program.

2. When giving any file path make sure to give the path from the root directory and refer to the example near the area you have to input.

3. Whenever the program refers to key it is referring to the Public key. The program will never refer anything to the Static key.

4. When outputting key to file make sure to not give any file extension as the program will automatically give the .skey extension to the file

Setting Up SAS-STE-Wizard

Downloading and Installing Dependencies

SAS-STE-Wizard needs JAVA JDK 17+ for it to work. So the first step would be to download and install JAVA JDK 17. If you already have JAVA JDK 17+ then you can move on to the next step.

Windows

1. If you are a Windows user you can download JAVA JDK 17 in page below. Open the link below: (https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html)
2. Then download the ‘ Windows x64 Installer ’
3. After that you can install it

Linux

1. If you are a Linux user you can download JAVA JDK 17 in the page below or download openjdk-17-jdk via apt. This tutorial will explain how to get it via apt (https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html)
2. Open up your terminal and type the following command
   $ sudo apt install openjdk-17-jdk
3. Once it finishes you are good to go

Downloading and Extracting SAS-STE-Wizard

1. You can download the SAS-STE-Wizard for Windows or Linux in the saaiqSAS official website (https://saaiqsas.github.io/tools/sas-ste.html) or in the SAS-STE GitHub repository (https://github.com/saaiqSAS/SAS-STE)
2. After downloading the zip file, extract it to your desire location, anywhere is fine

Setting up in Windows

1. After extracting SAS-STE-Wizard run/open the sas-ste.bat file in order to start the wizard.
2. That's all
3. If you are prompt with a dialog saying that it is not safe you can select run anyway as this program if totally safe. If you don't trust its safeness then you may read its source code yourself.

Setting up in Linux

1. After extracting SAS-STE-Wizard go into the directory extracted
2. Then open up a terminal in the directory and give sas-ste permission to be executable via the command below
   $ sudo chmod +x sas-ste
3. Then you can run the program via the following command
   $ ./sas-ste
4. That's all

Getting Started with SAS-STE-Wizard

The most basic way of using SAS-STE-Wizard is to encrypt pure plain multiple text with a generated key while outputting the result as pure plain text. Below are the steps to do it.

1. Type 1 (text) as input method
2. Type 2 (multiple times) as text input amount
3. Type 1 (encrypt) as function type
4. Type 2 (new key) to generate a new key
5. Type 1 to encrypt the data once
6. Type 1 (print key) to output key as pure plain text
7. Type 1 (print output) to output results as pure plain text
8. There you go, now you can type what ever you want to encrypt
9. You can refer to the ' commands: ' box for usable commands

If you want to send your encrypted text to a friend. copy the key starting from $ and to the final $ and send it to the friend. Then you can tell your friend to do the following steps below.

1. Type 1 (text) as input method
2. Type 2 (multiple times) as text input amount
3. Type 2 (decrypt) as function type
4. Type 1 (keyed) to use an existing key
5. Type 1 to input key as pure plain text
6. Copy and paste the key you sent as input
7. Type 1 (print key) to output key as pure plain text
8. Type 1 (print output) to output results as pure plain text
9. There you go, now you friend can copy and paste whatever encrypted text you send to decrypt it
10. Your friend can refer to the ' commands: ' box for usable commands

To change function types you can use the commands ' !FE! ' (Encrypt) and ' !FD! ' (Decrypt).

Using SAS-STE-Wizard at its max

Changing Static Keys

SAS-STE algorithm is actually designed to be used after changing Static keys. This way it gives more protection to your encrypted data. To change Static keys you can refer to the changing static keys segment under keys in SAS-STE segment

Encrypting Text as writing to a File

This allows you to type is text that will be directly saved to a file after being encrypted. To do this select multiple text input with a new key or existing key and select save to file as a output method.

Multiple Key encryption

SAS-STE allows you to encrypt your data all over again automatically based on your input after selecting new key. However, this way you will re encrypt your whole encrypted data with the same key.

If you want to use different keys then you can encrypt your data with one key and manually re encrypt the encrypted data with another key. This gives even more protection to your data.

So, when you want to decrypt the data you will have to manually decrypt the data twice by giving the keys in the correct order (reverse order or the keys outputted during encryption)

END OF DOCUMENT