**SECRET**

SocialCastr ID(SCID) Generation System

SECRET

*Preamble*

The SocialCastr ID (SCID) system is intended to provide a portable, reliable, anonymous, difficult to forge, unique, and easy-to-maintain identifier that users can use to verify their identities.

SCID became a necessity shortly after the release of version 0.91 beta of SocialCastr. It became apparent that because of the anonymous nature of the system, peers could not be assured of each other’s identities and so couldn’t reliably filter out the “noise”.

In effect, this meant never really knowing who you’re connected to, conversing with, exchanging files with, or any of the other features that SocialCastr currently has or may have in the future.

A channel, for example, can be completely customized so a unique identifier other than the channel name or description, for example, has to be employed. Alternately, peers otherwise have no way of being recognized on the broadcasting end either.

The SCID is currently implemented in ActionScript 3 but may be ported to other systems and languages in the future.

*Generating the SCID*

The SCID is generated the very first time that a user uses any of the SocialCastr software on their machine, after they’ve connected to the SocialCastr network.

Because of the security restrictions between AIR and web-based versions, an internal transfer mechanism (using LocalConnection, for example), can allow the SCID to be transferred between the various versions on the same machine. Allowing import via file or through any other external means (including websites, etc.), should be **strictly** disallowed in order to prevent tampering and SCID impersonation.

The SCID is generated using a combination of the local peer ID (the localPeerID property of the SwagCloud instance, as extracted from the NetGroup object). It’s typically a long, encrypted hexadecimal string of SHA-256 hashed data such as: 569c480c84aa4039cefdb5eb35ac7bed85b4048caf7a4b03a785a4157ea62ecf

Because this value is required, an initial connection to SocialCastr’s Announce group must first be established (easily accomplished via the Announce class).

The string is treated as a linear sequence of hex value pairs. That is, using the above example ID, the first hex value is 56, then 9C, then 48, and so on.

The output buffer should be binary-friendly like a ByteArray or an array of uints. It will be required to hold arbitrary binary data until it can be down-sampled to readable text.

The first hex value of the string is used for the encoding key for the remainder of the string (String-Encoding Key, or SEK), and the final hex value is the encoding key for the SEK (Key-Encoding Key, or KEK).

The first step in the encoding process requires reading the SEK and KEK. Convert each valid hex value to a uint value (unsigned integers are safer since the sign bit isn’t used).

After this, the first hex value in the string is encoded in the following way: the first value (only) is XORed with the SEK, then KEK, and finally the hex value AA. This obfuscates the key for the remainder of the string, which is transformed simply by XORing each value with the SEK.

Note that the SEK is the original value of the first hex value of the string, not the encoded value.

Finally, a date/time stamp is appended to the buffer as numeric values in the following format:

**ddMMyyHHmmSSms**

The “dd” value represents the day of the month (1-31), “MM” is the month (1-12), “yy” is the full four-digit millennial year, “HH” is the hours value of the current system clock (0-23), “mm” is the minutes value of the current system clock (0-59), “SS” is the seconds value of the system clock, and “ms” is the milliseconds value of the clock.

The output buffer should now be exactly half the length of the original **localPeerID** value, plus 7 additional bytes for the date stamp.

Once all of the encoded values are stored in the output buffer array, two additional operations must be performed on them to make them more human-friendly.

The first step to making the buffer friendlier is to apply a modulo 26 shift (decimal) on each value in the output buffer (to push the values into a valid letter range), then add 65 decimal to each value. Combined, this operation reduces the resolution of the data, and then puts it into the range of the ASCII range for the letters “A” to “Z”.

**Example**

Using the local peer ID above as an example, the initial conversion of hex to decimal values would produce:

86 156 72 12 132 170 64 57 206 253 181 235 53 172 123 237 133 180 4 140 175 122 75 3 167 133 164 21 126 166 46 207 dd MM yy HH mm SS ms

Note that the final date / time values are to be determined when the SCID is being generated, so they’re represented as variables.

The first hex value is created by XORing 86 with 207 (the KEK) and AAh, which produces **51**. So the SEK is 86, the KEK is 207, and the first encoded value is 51.

After this, each value is simply XORed with the SEK (86), to produce:

This produces:

51 202 30 90 210 252 22 111 152 171 227 189 99 250 45 187 211 226 82 218 249 44 29 85 241 211 242 67 40 240 120 153 …

The date/time stamp has been omitted in this example, but don’t forget to add it to the real value!

Since some of these values are outside of the printable ASCII range, a modulo 26 operation is applied on all of them. This produces:

25 20 4 12 2 18 22 7 22 15 19 7 21 16 19 5 3 18 4 10 15 18 3 7 7 3 8 15 14 6 16 23 …

Finally, although these values are in the range 1 to 26, they still don’t fit into the window of ASCII characters in the range “A” to “Z”, so the final operation adds 65 to every value in the string. This produces:

90 85 69 77 67 83 87 72 87 80 84 72 86 81 84 70 68 83 69 75 80 83 68 72 72 68 73 80 79 71 81 88 …

Now each value can be converted directly to an ASCII character to produce:

ZUEMCSWHWPTHVQTFDSEKPSDHHDIPOGQX …

In practice, this will also require the date/time stamp appended to the end. Using the following values:

hours=14  
minutes=18  
seconds=43  
milliseconds=807  
date=24  
month=1  
year=2012  
  
… the resulting encoded string would be:

ZUEMCSWHWPTHVQTFDSEKPSDHHDIPOGQXIIIKIIIQIIIVIIHJIIIAIIIJIIDI

A valid SCID should always be 60 characters long:

64 characters in localPeerID divided by 2 = 32

Each date value is 4 bytes (32 bits) = 7 \* 4 = 28

Total length = 60

Note that the “0” values in the date (converted to “I”), can be filled with random characters if desired. The aim of the SCID is to produce an anonymous, 60-character-based, unique identifier, and as such, other than the character range and string length, no validation can be expected to take place.

The generation system can therefore be tweaked to a certain extent.

*Storing the SCID*

The SCID should be stored in an area and in a way as to make it extremely difficult to tamper with.

In Adobe AIR this should be accomplished via encrypted local storage:  
<http://help.adobe.com/en_US/air/html/dev/WS5b3ccc516d4fbf351e63e3d118666ade46-7e31.html>

In the web version it may be stored in the Local Shared Object (unless a more secure mechanism is discovered):  
<http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/SharedObject.html>

In the local shared object, the shared object name is “\_\_$¢1D “ (the additional character at the end is ASCII 255, and the variable name for the stored SCID is “\_\_scratchdisk”.

Regardless of the method used, obfuscation should be used where possible (don’t use obvious naming conventions), and the process of generating or storing the SCID should *never* be displayed in any debug or output panel traces.