## Triple DES and AES 192/256 Implementation Notes

## Sample Password-to-Key and KeyChange results of Triple DES and AES 192/256 implementation

## For InterWorking Labs customers who require detailed information on the implementation of the AES 192/256 and 3DES in SilverCreek, the SNMP Test Suite

## 1. TripleDES passwordToKey and keyChange samples

Algorithm description:
http://tools.ietf.org/html/draft-reeder-snmpv3-usm-3desede-00
Chaining of the Password-to-Key Algorithm

Some cryptographic algorithms may require keys that have a length greater than the that of the hash output used by the password-to-key algorithm. This will be the case, for example, with any user that defines usm3DESEDEPrivProtocol as its privacy protocol (described below in Section 6). To acquire the necessary number of key bits, the password-to-key algorithm may be chained using its own output as further input in order to generate an appropriate number of key bits.

Chaining is described as follows. First, run the password-to-key algorithm with inputs of the passphrase and engineID as described in the USM document. This will output as many key bits as the hash algorithm used to implement the password-to-key algorithm. Secondly, run the password-to-key algorithm again with the previous output (instead of the
passphrase) and the same engineID as inputs. Repeat this process as many times as necessary in order to generate the minimum number of key bits for the chosen privacy protocol. The outputs of each execution are concatenated into a single string of key bits.

When this process results in more key bits than are necessary, only the most significant bits of the string should be used.

For example, if password-to-key implemented with SHA creates a 40-octet string string for use as key bits, only the first 32 octets will be used for usm3DESEDEPrivProtocol.

Chaining may be demonstrated using simplified pseudo-code as follows, let:
Output_bits <-- P2K( Input_bits, EngineID )
where the string of key bits (Output_bits) is returned from the password-to-key (P2K) algorithm which takes a string of bits (Input_bits) and the engineID (EngineID) as inputs. One iteration of chaining, creating a localized key of twice the normal length is achieved as follows:

```
K1 <-- P2K( <passphrase>, <engine_id> )
K2 <-- P2K( K1, <engine_id> )
localized_key = K1 | K2
```

The next further iteration will pass K2 (instead of K1) and return K3. The iteration after that passes K3 and returns K4, etc. The results of all iterations (K1, K2, .., Kn) are concatenated to form the localized key. Note that the engineID is the same for all iterations.

A TripleDES 32 octets (256 bits) keys with MD5

## A.1. Password-to-Key Chaining Sample Results using MD5

The following shows a sample output of the password-to-key algorithm for a 32-octet key using MD5. The password used in this example is "maplesyrup". The first 16 octets (bytes 1 through 16) are
generated by the password-to-key algorithm with the password as input. The second 16 octets (bytes 17 through 32) are generated from the password-to- key algorithm with the first 16 octets as input.

Each invocation of the password-to-key algorithm in the generation of a string of key bits uses the same engineID. In this example the engineID is:
'00 0000000000000000000002 'H

The final output of the password-to-key algorithm, used twice as described above, produces a 32-octet localized key of: '52 6f 5e ed 9f cc e2 6f 8964 c2 930787 d8 2b 79 ef f4 4a 90650 e e0 a3 a4 Oa bf ac 5a cc 12'H

A 2. Sample keyChange Results for 32-octet keys

Sample keyChange Results for 32-octet Keys Using MD5

Let us assume that a user has a current password of "maplesyrup" as in section A.1. and let us also assume an snmpEngineID of 12 octets:
'00 00000000000000000000 02'H

If we now want to change the password to "newsyrup", then we first calculate the localized key for the new password. It is as follows:

8702 1d 7b d9 d1 01 ba 05 ea 6e 3b f9 d9 bd 4a
7029 8b 75 7c 9199 b6 a8 fb f3 93 7b e0 54 86'H

Then, using the following value as a placeholder for the random value:

$$
\begin{aligned}
& \text { '00 } 000000000000000000000000000000 \\
& 00000000000000000000000000000000 \text { 'H }
\end{aligned}
$$

We compute a keyChange value of:
'00 000000000000000000000000000000
00000000000000000000000000000000
ac fb 190 e d2 e2 2950 f5 7a cd 9068 cb 1e b2
a5 56 cc eb c8 f3 ba d0 c4 055157 db a8 d2 6b'H

B TripleDES 32 octets (256 bits) keys with SHA

## B.1. Password-to-Key Chaining Sample Results using SHA

The following shows a sample output of the password-to-key algorithm for a 40-octet key using SHA. The password used in this example is "maplesyrup". The first 20 octets (bytes 1 through 20) are generated by the password-to-key algorithm with the password as input. The second 20 octets (bytes 21 through 40) are generated from the password-to- key algorithm with the first 20 octets as input. Each invocation of the password-to-key algorithm in the generation of a string of key bits uses the same engineID. In this example the engineID is:
'00 00000000000000000000 02'H

The final output of the password-to-key algorithm, used twice as described above, produces a 40-octet localized key of:
'66 95 fe bc 9288 e3 $6282235 f$ c7 $151 f 128497$ b3 $8 f$ 3f
9b 8b 6d 7893 6b a6 e7 d1 9d fd 9c d2 d5 06554774 3f b5'H
B.2. Sample keyChange Results for 32-octet Keys Using SHA

Let us assume that a user has a current password of "maplesyrup" as in section B.2. and let us also assume the snmpEngineID of 12 octets:
'00 00000000000000000000 02'H

If we now want to change the password to "newsyrup", then we first calculate the localized key for the new password. It is as follows:

78 e2 dc ce 79 d5 9403 b5 8c 1b ba a5 bf f4 63
91 f1 cd 2597743555 f9 fc f9 4a c3 e7 e9 22'H

Note that this value has been truncated from 40 to 32 octets.
Then, using the following value as a placeholder for the random value:
'00 000000000000000000000000000000
00000000000000000000000000000000 'H
we compute a keyChange value of:
'00 000000000000000000000000000000
00000000000000000000000000000000
ce 1328 fb 9a 9c 19 ce c151 a3 5a 77 f9 2039
ca ff 00 c9 b3 9b 19 a0 5e 0175559437 6a 57'H

## 2. AES192 and AES256 passwordToKey and keyChange samples

Algorithm description:
http://tools.ietf.org/html/draft-blumenthal-aes-usm-04
Short Localized Keys

The encryption protocols defined on this memo SHOULD be used with an authentication protocol that generates a localized key with enough key material to derive a 128/192/256 bits encryption key. At the time of this writing an authentication protocol with such characteristics has not been defined within the USM model for the SNMPv3 architecture.

However, if the size of the localized key is not large enough to generate an encryption key the following algorithm is applied to extend the localized key:
1)Let Hnnn() the hash function of the authentication protocol for the user $U$ on the SNMP authoritative engine E. nnn being the size of the output of the hash function (e.g. nnn=128 bits for MD5, or nnn=160 bits for SHA1).
2)Set c = ceil ( $256 / n n n)$
3)For i $=1,2, \ldots, \mathrm{c}$
a.Set Kul = Kul || Hnnn(Kul); Where Hnnn() is the hash
function of the authentication protocol defined for that user

As an example if the user authentication protocol is HMAC-SHA1-96, the hash function Hnnn is SHA1 with nnn=160 bits. The algorithm will generate a localized key 480-bit long:
Kul' = Kul || SHA1(Kul) || SHA1(Kul||SHA1(Kul))
A. 24 or 32 octets keys with MD5

## A.1. Password-to-Key Chaining Sample Results using MD5

The following shows a sample output of the password-to-key algorithm for a 24-octet or 32-octet key using MD5. The password used in this example is "maplesyrup". The first 16 octets (bytes 1 through 16) are generated by the password-to-key algorithm with the password as input. The second 8 octets (bytes 17 through 32) are generated from the MD5 hash function (NOT the password-to- key algorithm as used in Triple DES) with the first 16 octets as input.

The invocation of the password-to-key algorithm in the generation of a string of key bits uses an engineID. In this example the engineID is:

The final output of the password-to-key algorithm, used as described above, produces a 32-octet localized key for AES256:
'52 6f 5e ed 9f cc e2 6f 8964 c2 930787 d8 2b
fa 24 a9 246742 6c 2f 4b 0919 2b e1 Od fa ec'
or a 24-octet localized key for AES192:
'52 6f 5e ed 9f cc e2 6f 8964 c2 930787 d8 2b
fa 24 a9 246742 6c 2f'
A.2. Sample keyChange Results for 32-octet or 24-octet keys

Sample keyChange Results for 32-octet Keys Using SHA

Let us assume that a user has a current password of "maplesyrup" as in section C.1. and let us also assume an snmpEngineID of 12 octets:
'00 0000000000000000000002 'H

If we now want to change the password to "newsyrup", then we first calculate the localized key for the new password. It is as follows:
--32-octet AES256 key--
8702 1d 7b d9 d1 01 ba 05 ea 6e 3b f9 d9 bd 4a
Od ad 14 1a f6 d8 0371 b5 b3 cc de a9 8311 c4'
--24-octet AES256 key--
8702 1d 7b d9 d1 01 ba 05 ea 6e 3b f9 d9 bd 4a
Od ad 14 1a f6 d8 03 71'

Then, using the following value as a placeholder for the random value:

```
--32-octet AES256 keyChange--
    '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'H
--24-octet AES192 keyChange--
    '00 00 00 00 00 00 00 00 00 00 00 0000000000
    O0 00 00 00 00 00 00 00'H
```

we compute a keyChange value of:
--32-octet AES256 keyChange--
'00 000000000000000000000000000000 00000000000000000000000000000000 $4 f 86$ e0 ad 0335 be 93 6e 0d 0b $004 d$ a8 8e 36 c1 d9 57 1d 6a a2 Of 4097 a3 f3 a7 3d cf 44 ba'
--24-octet AES192 keyChange--
'00 000000000000000000000000000000 0000000000000000547281 da 7d 4f f5 80 aa 5c d5 85 ab b6 5f e6 6877 6b 63 5b 45 c7 b4'
B. 24 or 32 byte keys with SHA
B.1. Password-to-Key Chaining Sample Results using SHA

The following shows a sample output of the password-to-key algorithm for a 24-octet or 32-octet key using SHA. The password used in this example is "maplesyrup". The first 20 octets (bytes 1 through 20) are generated by the password-to-key algorithm with the password as input. The second 8 octets (bytes 21 through 40) are generated from the SHA hash function (NOT the password-to- key algorithm
as used in Triple DES) with the first 20 octets as input.
The invocation of the password-to-key algorithm in the generation of a string of key bits uses an engineID. In this example the engineID is:
'00 00000000000000000000 02'H

The final output of the password-to-key algorithm, used as described above, produces a 32-octet localized key for AES256:
'66 95 fe bc 9288 e3 $6282235 f$ c7 15 1f 1284
97 b3 8f 3f 505 e 07 eb 9 a f2 5568 fa 1f 5d be'
or a 24-octet localized key for AES192:
'66 95 fe bc 9288 e3 $6282235 f$ c7 15 1f 1284
97 b3 8f 3f $505 \mathrm{e} 07 \mathrm{eb}^{\prime}$
B.2. Sample keyChange Results for 32-octet or 24-octet keys

Sample keyChange Results for 32-octet Keys Using SHA

Let us assume that a user has a current password of "maplesyrup" as in section D.1. and let us also assume an snmpEngineID of 12 octets:
'00 00000000000000000000 02'H

If we now want to change the password to "newsyrup", then we first calculate the localized key for the new password. It is as follows:
--32-octet AES256 key--
78 e2 dc ce 79 d5 9403 b5 8c 1b ba a5 bf f4 63
91 f1 cd 25 f7 8279 f8 0632 dd e1 1c f5 9e 25
--24-octet AES256 key--
78 e2 dc ce 79 d5 9403 b5 8c 1b ba a5 bf f4 63 $91 \mathrm{f1}$ cd 25 f7 8279 f8'

Then, using the following value as a placeholder for the random value:
--32-octet AES192 keyChange-'00 000000000000000000000000000000 00000000000000000000000000000000 'H
--24-octet AES192 keyChange-'00 000000000000000000000000000000 00000000000000 00'H
we compute a keyChange value of:
--32-octet AES256 keyChange--
'00 000000000000000000000000000000
00000000000000000000000000000000
00 f9 of 0c 9e 4560 7d 5c cb 0c 3a d5 60 a7 76
dc 7020 a2 bb 8104 d7 20 e d2 b2 ac 8917 7c
--24-octet AES192 keyChange--
'00 000000000000000000000000000000
0000000000000000 d2 a1 7d 8f 3c ce c6 49
da 838845 e7 7f 61 a9 b7 bb 9a 20 f6 3f 2f 89'

## Need a justification to purchase?



