50 Million Keys to SNMPv3 Privacy

A Technical Paper prepared for SCTE•ISBE by

Paul E. Schauer
Distinguished Engineer
Comcast Cable
183 Inverness Drive West, Englewood, CO, 80111, USA
+1-303-372-1215
paul_schauer@comcast.com

Table of Contents

Title **Page Number**

List of Figures

Title **Page Number**

1. Introduction

Security and Privacy must be top line features of any service operating today. Twenty years ago, the team at CableLabs prescribed SNMPv3 with Diffie Hellman key exchange for encrypting management traffic from DOCSIS cable modems. While SNMP is dated, simple non-sensitive management data is still integral to CM operations. Yet in 2020, few production implementations of encrypting SNMP have been reported. Because CM management traffic lives behind numerous other layers of network and physical security, this has not been a significant issue. Still, leaving the CM management data in clear text SNMP is an operational luxury that should be phased out of practice. Switching to newer, more secure protocols is a future solution that overlooks the millions of installed DOCSIS CMs. As part of Comcast’s ongoing evolution of security and privacy, clear text SNMP data has been deprecated. DOCSIS cable modem management is now encrypted with SNMPv3 utilizing Diffie Hellman key exchange as specified by CableLabs. Associated mechanisms for securely managing the privacy keys and bootfiles are part of the larger solution. This paper will highlight technical issues of implementing SNMPv3 with Diffie Hellman key exchange at MSO scale.

2. Don’t Panic

The information exchanged over SNMP is focused on operating network equipment. The information includes data such as the uptime of the device,. the list of interfaces and their name, and the number of packets going in and out of the device. There should not be any sensitive material such as credit card numbers, account numbers, or other identifiers contained in the data. Since the first two versions of the protocol had no security, the optimistic expectation for implementors was “don’t put sensitive data into SNMP MIBs”.

This understanding may have been adequate in the 1980s, when SNMP was first created. In 2020, this is no longer tenable. Various SNMP-based denial of service attacks pose new threats. In addition, privacy requirements that have evolved make this a protocol that should be removed from modern networks.

There are hundreds of millions of DOCSIS cable modems providing service in cable operators’ networks that use SNMP. Replacing all of them to remove SNMP is operationally unrealistic.

This is where adding encrypted SNMPv3 with Diffie Hellman key exchange provides an important layer of privacy for these devices. CableLabs and the IETF released RFC 2786 in 2000, and DOCSIS cable modems are required by the CableLabs DOCSIS CM-SP-OSSI specification to implement the RFC.

Actually deploying DH key exchange at scale was left to implementers: “The configuration [of the cable modem SNMP agent] would be done either manually (in the case of a small number of devices), or via some sort of distributed configuration file. The actual mechanism is outside the scope of this document [RFC 2786].”

This document provides a guide for that out of scope effort. This document does not reveal Comcast’s implementation of an SNMPv3 DH key management system. There are decision points and examples for needed subsystems that can constitute an overall solution. Any given implementation will be unique to the environment where it is deployed.

All examples are for illustration purposes only and should not be used in actual production without formal review.

3. Get On With It

Key design decision: Get over the debate and get on with the deployment of DOCSIS CM SNMPv3 with DH key exchange.

The MD5 hashing algorithm and the DES encryption algorithm with 56-bit keys prescribed in RFC 2786 have been deprecated. They are obsolete because they can be compromised. The SNMP User Security Model described in RFC 3414 highlights the limitations of the overall SNMP security model.

If the information from the cable modems has a low security value, and it is protected by many other layers of security, why even bother obscuring it with known weak encryption?

This question may be argued numerous times and numerous ways by numerous engineers. They will raise valid technical points. Undertaking the journey to SNMPv3 with DH key exchange starts with managerial direction, not technical requirements. For instance, there may be a corporate directive to remove SNMPv2 in the network. A corporate information policy may require that systems encrypt all data in transit. The key here is to stop the debate and get on with implementation.

4. Cue the Diffie Hellman Key Exchange

Key design decision: Validate whether your code base supports the use of local keys for the SNMP agent on the cable modem.

As of this writing, the Net-SNMP utilities and Java's SNMP4j library support the use of local keys. This section summarizes differences between SNMPv2 community, SNMPv3 USM, and SNMPv3 USM with DH key exchange. It uses the Net-SNMP command line tools to demonstrate these differences in the calls to each service. This knowledge is required to plan necessary upgrades to the current SNMP polling software. An SNMP poller is formally an "SNMP Command Generator" or "SNMP manager" and can be implemented in numerous forms.

4.1. SNMPv2 Community

Reinforcing that "S stands for Simple", the SNMPv2 community string is a plain text way to group SNMP managers' and agents' commands and responses. The community string is sent in clear text as part of an SNMPv1 or v2 packet. This offers no security at all nor was it intended to.

```
$ snmpget -v 2c -c commString 10.168.6.82 system.sysUpTime.0
10.53.115.71.58645 > 10.168.6.82.161:  { SNMPv2c C="commString"
{ GetRequest(28) R=445233587  .1.3.6.1.2.1.1.3.0 } }
10.168.6.82.161 > 10.53.115.71.58645:  { SNMPv2c C="commString"
{ GetResponse(32) R=445233587  .1.3.6.1.2.1.1.3.0=182774513 } }
```

Figure 1 - SNMPv2 Community Example

4.2. SNMPv3 User-based Security Model with no authentication or privacy

This may seem counter-intuitive, but SNMPv3 provides a method for unsecured communication. The SNMPv2 "community" evolves into the SNMPv3 "user". There is a default user with default views and permissions prescribed in RFC 2786 that all DOCSIS cable modems implement. This becomes important later on for exchanging keys. This example uses the default DOCSIS user "dhKickstart" with no authentication or privacy keys in the request.

```
$ snmpget -v 3 -l noAuthNoPriv -u dhKickstart 10.168.6.82
system.sysUpTime.0
```

Figure 2 - SNMPv3 No Authentication No Privacy Example

4.3. SNMPv3 User-based Security Model with authentication and privacy

SNMPv3 adds methods for authenticating message transmission and reception. It also provides methods for encrypting the contents of the message in transit. The standard RFC 3414 implementation uses an authentication key and a privacy key that are loaded into the SNMP manager and the SNMP agent(s). The manager and agents use these keys in the algorithms prescribed by RFC 3414 to transmit and decrypt messages in lieu of plain text SNMPv2.

```
$ snmpget -v 3 -l authPriv -u v3keysTest -A TestAuthenticationKey -X
TestPrivacyKey 10.168.6.21 system.sysUpTime.0
```

Figure 3 - SNMPv3 Authentication + Privacy Example

4.4. SNMPv3 USM authentication and privacy with DH key exchange

The addition of DH key exchange prescribes a method where the authentication and privacy keys are derived based on cryptographic methods. This improves on the standard SNMPv3 implementation because it removes the actual keys from the cable modem configuration. This example shows authentication and privacy keys that are precomputed based on the DH key exchange, and are then used in the call to the Net-SNMP utilities.

```
$ snmpget -v 3 -l authPriv -u v3localKeys -3k
0xf9c96a9232ee65a08aa9085e6f1ff82c -3K
0xedebac85112645218fb7a63659a332c2 10.168.6.21 system.sysUpTime.0
```

Figure 4 - SNMPv3 Authentication + Privacy + Local Keys Example

These examples are simple demonstrations of how calls from an SNMP manager to an SNMP agent change. The specific code used in a cable provider's system will need to be upgraded similarly.

5. Roll a Random Number

Key design decision: Creating and managing the distribution of the Manager Random number(s).

The manager random number is the basis for all subsequent activities associated with RFC 2786 DH key exchange. A random number is transformed according to the algorithms in RFC 2786 to arrive at the manager public key that must be included in the cable modem boot file. Each calculated public key is exposed through the CM's SNMP MIB as a "usmDHKickstartMgrPublic" key. Each SNMP user profile must have its own manager random number to create its unique key. The cable modem follows a similar process and creates its own "usmDHKickstartMyPublic" key for each user as well. These values are then further processed and used in calls from the SNMP manager to the SNMP agent as shown in the previous section.

The simplest and least secure way to manage the random number is to make it available to the SNMP polling systems the same way the SNMPv2 community string was provided. Methods such as email, text files, or hard coded configuration pushes allow any system to calculate the DH keys based on the manager random number. This is also the least secure method a system designer could choose.

More secure solutions also require more effort to manage. Code repositories or secured APIs with token authentication schemes can be used. A formal key vault or similar key management infrastructure may be used to authorize access to a given manager random number for a given user.

Since an SNMP manager needs to be able to calculate the authentication and privacy keys any time a cable modem reboots, the scaling of the system which manages needs to be carefully considered and expanded to accommodate the potential query traffic. The parameters a system designer needs to consider include: the number of SNMP USM profiles with unique manager random numbers, the number of different boot files in the system, and the frequency of change of the manager random numbers. Since an SNMP manager could calculate the keys itself, the number of cable modem reboots in the overall system is less important at the manager random number distribution level.

The system designer may also wish to avoid handing out the manager random number(s) completely, and instead offer a service that provides the pre-calculated keys to requestors. This keeps the manager random numbers from being cached or redistributed to other parties.

6. Keeper of the Keys

Key design decision: Manage the distribution of calculated USM keys in lieu of sending out manager random numbers

Another layer of flexibility can be added with a USM key management service. Instead of directly sending out the manager random numbers, a dedicated service can be created to produce the authentication and privacy keys to requestors.

There are security benefits to a USM key distribution service. The manager random numbers are not divulged to the requesting SNMP manager, only the calculated keys. This improves the security of the overall system. It also removes the necessity of SNMP managers to calculate the keys themselves, which may require significant code updates.

There are also drawbacks to distributing calculated keys. The primary challenge is scalability. Since the cable modem public keys change every time the cable modem reboots, the service must be able to refresh the CM public key from any modem when it reboots. A separate key distribution also creates a serial failure scenario. If SNMP managers must retrieve the precomputed keys from a key distribution service, and the service is impaired or down, visibility will be lost to the CMs as they reboot over time.

A separate USM key service like this may be used for all the users and all the keys, or it may be selective for only the user(s) that have read and write access. This may provide a compromise between potential loss of visibility, query rate, and key integrity.

The final consideration for a USM key service is the potential query rate. The CM keys change every time they reboot, hence the USM service must retrieve and recalculate the keys each time. Then the new keys must be made available to any SNMP manager using the service. The number of cable modems, the number of reboots, the number of SNMP managers in the system, and the frequency of SNMP queries from the SNMP manager are all scaling factors that will result in the potential query rate the key store would need to accommodate.

7. Pick User Profiles

Key design decision: SNMP users and roles to include in the CM bootfile to generate DH keys. Each user profile will create its own public key based on its own manager random number, which will then be used to calculate the local keys for that user.

The unsecured "dhKickstart" user documented in RFC 2786 has a basic view and access to the public keys necessary to calculate the actual CM local keys. Additional users should be added with appropriate views and permissions to monitor the cable modems. Operators may leverage existing RFC 3414 USM and RFC 3415 VACM profiles in their cable modem boot files. These may be copied or further refined as desired for SNMPv3 with DH key exchange.

There is nothing extraordinary about these users or profiles. The one decision point of note is whether to partition a read/write user and read/only users.

8. Break Out the Bootfiles

Key design decision: How the system will manage the bootfiles with the new users and keys.

Cable modem provisioning systems already manage multiple bootfiles. Adding the SNMPv3 users and manager random numbers to the bootfiles may increase this quantity. There are numerous ways for a cable operator to partition the bootfiles. Groupings might include the CM manufacturer, model number, physical location, organizational responsibility, provisioning complex, or other criteria.

Dynamically generated bootfiles with a unique manager random number for each user on each cable modem would be the most secure solution. If an actor obtained one manager random number for a given CM, an SNMP session could be negotiated to only that cable modem with the specific user. The manager random number would change when that CM rebooted, and all the security parameters would be reset. This is also the most difficult scenario to manage. The provisioning system that generates the bootfiles would also have to seed the distribution system for all the manager random number(s) as described above.

A single random manager number that is used for the keys in every bootfile is the easiest solution, and clearly the least secure. Any actor that obtains the manager random number could securely negotiate DH sessions with any cable modem in that system.

Deciding how many users and manager random numbers could thus simply leverage the existing bootfile management method. It could also completely change the management and distribution of them.

9. Double Click

Key design decision: Whether SNMP pollers rely on cached keys, or query and calculate keys for each session.

As shown below, querying the usmDHKickstartMyPublic key(s) from a given cable modem with the default dhKickstart user should always return the cable modem public keys.

```
$ snmpbulkwalk -v 3 -m ALL -l noAuthNoPriv -u dhKickstart 10.168.6.82.161
snmpUsmDHObjectsMIB
SNMP-USM-DH-OBJECTS-MIB::usmDHKickstartMyPublic.1 = Hex-STRING:
97 BC 5A C7 B8 FC 32 80 07 2E 20 4A CE 6D 59 86
0E 0B C8 B2 F7 DD 86 4F 7D B3 E6 21 B2 51 99 32
FE 62 9A 20 B3 CD 72 97 5A 18 B5 E1 87 02 89 AB
6B 67 9F 38 1C BA E4 07 A8 BA 9D 80 40 BE 3B 26
C6 B9 9E F8 D3 70 0E A3 3A 34 95 3F 51 A2 55 95
EC AE FF 84 A9 72 57 8C AB 36 45 76 8B 7F 32 95
C6 BD 93 D9 DB CF 6A 2E 05 10 CE 9E 2E 4A 01 CB
1B 27 42 68 25 BD 54 55 79 79 93 AE 8C 61 47 AB
```

Figure 5 - CM Public Key Query Example

Whether the SNMP manager is a subsystem within a USM key management service described previously or a separate manager that has access to the SNMP manager random number for a given user, the fact that the keys only change when the cable modem reboots can be used as a convenience to avoid double querying the modems.

10. Prime the Pollers

Key design decision: Scaling the SNMP poller inventory to accommodate the key exchange and encryption overhead.

The additional response time from the CMs with the key exchange and encryption at MSO scale is non-trivial yet manageable. Response time measurements of agent response in laboratory settings have shown small increases of a few milliseconds or less. Scaling out to CMTSs with thousands of CMs and carrying full customer traffic have shown increases from a few milliseconds to a few seconds depending on CM and CMTS load at the time. Some phenomena observed in the production network include:

“Warm up failures”, or CM key exchanges that take longer than expected.

Serialized key negotiation to sets of modems that should be converted to parallel queries.

Data response times that take randomly longer than the SNMPv2 equivalent.

The solution for a system designer is the same: additional SNMP pollers with code optimized for parallelism. Experience at Comcast’s scale shows a 30% to 300% increase in resource needs depending on the polling software and computing platform. Physical servers, virtual containers, threading models, cloud computing instances, and more may need to be scaled specifically to meet the overall system demands. There has not been a universal solution observed at Comcast’s scale; all of the above have needed some adjustment based on full production deployment. The good news is they were all identified during the roll out and addressed so that the project was deployed successfully.

11. Conclusion

Adding privacy by encrypting cable modem management traffic starts by opening the door to RFC 2786. The keys in this presentation are provided to help guide design decisions necessary to implement SNMPv3 with DH key exchange on CMs at production scale. The balance between maintainability and scalability of each key must be considered as part of the overall system design and implementation.

Abbreviations

CM	cable modem
DH	Diffie Hellman
DOCSIS	data over cable service interface specification
PBKD	password based key derivation
RFC	request for comments
SNMP	simple network management protocol
SCTE	Society of Cable Telecommunications Engineers
USM	user security model
VACM	view-based access control module

Bibliography & References

RFC 2786: *Diffie-Helman USM Key Management Information Base and Textual Convention*; RFC Editor

RFC 3414: *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*; RFC Editor

RFC 3415: *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)*; RFC Editor

Appendix 1 RFC 2786 DH key exchange simulation

All of the constants and mathematics associated with the DH key exchange are documented in RFC 2786. The following illustrates these calculations using open source tools. The methods shown are long, single steps for demonstration purposes. Actual systems implementation would be succinct.

Pick a 1024 bit manager random number:

```
$ python3 -c 'import secrets; print( hex ( secrets.randbits(1024) ) )'
```

```
0x8ad7112ed1742765da233bb761f44cc69b329161e1a2a1c28032c25445566284bf6ca06a8a91b2a49c06
b7198aeb1574570b1e1ab22f7a9c471d01ed1f260f7356e3c88b92f06e70a14ac91480677aa4c571fb0d1c
f0a9857d027f98b1426701ab6fb27a933b0db3821b7c401fdcb7f4077ae87c4f64f682a0ffe54ff5c90928
```

Then transform the random number as prescribed using the prime number listed in the RFC The equation to derive the keys is: (((base number) ^ random number) modulo DH prime number):

```
$ python3 -c 'print( hex( pow (2,
0x8ad7112ed1742765da233bb761f44cc69b329161e1a2a1c28032c25445566284bf6ca06a8a91b2a49c06
b7198aeb1574570b1e1ab22f7a9c471d01ed1f260f7356e3c88b92f06e70a14ac91480677aa4c571fb0d1c
f0a9857d027f98b1426701ab6fb27a933b0db3821b7c401fdcb7f4077ae87c4f64f682a0ffe54ff5c90928
,
0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A
08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A6
37ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE65381FFFFFFFFFFFFFFFF
) ) )'
```

The result is a “usmDHKickstartMgrPublic” number that would then be included in the CM bootfile and exposed by the CM SNMP MIB:

```
0xef26516697b54859842f6fdc933a9aeac3750a1d50472808e62f845efb604caf5e97b117b061538a057b
1fef375da865cc0344d36f7cc6e0cc7265aa439f423d250c20518ba8459df50d710ba30a4292d43b7fbcee
189faa12cafdafd54c4797aac2bdb5438b5f7a8a5df28b12673b95a8a17682361815fd026960bb5ab021dc
```

Internally, the cable modem follows the same algorithm each time it reboots:

```
$ python3 -c 'import secrets; print ( hex ( secrets.randbits(1024) ) )'
```

```
0x94815e16cb9ac643050278ef80825a384f946d3c050228c440700a7cfd7b4f4acc58b095d51be81ec11e
d5129434588384e771d98d327282fc9d4ff1b81930563a0d5f988952f6b166cbcd6bb692d8c1b2c320762d
2b24d5eea0a0fde537424f2fc09f8a1d65539d26472bc45b60bbc2ea03d2fa98b996c8677e0419c4fe825d
```

```
$ python3 -c 'print ( hex ( pow (2,
0x94815e16cb9ac643050278ef80825a384f946d3c050228c440700a7cfd7b4f4acc58b095d51be81ec11e
d5129434588384e771d98d327282fc9d4ff1b81930563a0d5f988952f6b166cbcd6bb692d8c1b2c320762d
2b24d5eea0a0fde537424f2fc09f8a1d65539d26472bc45b60bbc2ea03d2fa98b996c8677e0419c4fe825d
,
0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A
08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A6
37ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE65381FFFFFFFFFFFFFFFF
) ) ) '
```

The result is a simulated “usmDHKickstartMyPublic” key that would be exposed by the CM:

```
0x992d538d0cf24e4869b3cdc8e04f7ee6c6fb454b2b7b412a585dc09d9c293167cea9af58710695ff9344
ca7770da876f3124034af013933d257cddc21930886b364dcad417d8af56ed8b7b6953e7de5f8f0ad3c5a1
5f85578d9a64a7900bb58646c8bf804dc099da5428d1308db33c4cab828ca1618e2258561ba66f9c23ad3
```

Now that both the CM and the SNMP manager have known public keys, each calculates a shared secret using the other's public key:

At the SNMP manager:

pow (`usmDHKickstartMyPublic, manager random number, DH prime)`

```
$ python3 -c 'print ( hex ( pow
( 0x992d538d0cf24e4869b3cdc8e04f7ee6c6fb454b2b7b412a585dc09d9c293167cea9af58710695ff93
44ca7770da876f3124034af013933d257cddc21930886b364dcad417d8af56ed8b7b6953e7de5f8f0ad3c5
a15f85578d9a64a7900bb58646c8bf804dc099da5428d1308db33c4cab828ca1618e2258561ba66f9c23ad
3,
0x8ad7112ed1742765da233bb761f44cc69b329161e1a2a1c28032c25445566284bf6ca06a8a91b2a49c06
b7198aeb1574570b1e1ab22f7a9c471d01ed1f260f7356e3c88b92f06e70a14ac91480677aa4c571fb0d1c
f0a9857d027f98b1426701ab6fb27a933b0db3821b7c401fdcb7f4077ae87c4f64f682a0ffe54ff5c90928
,
0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A
08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A6
37ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE65381FFFFFFFFFFFFFFFF
) ) )'
```

Shared secret at the SNMP manager:

```
0xf4729f7ec7f55e997060e85688b72efa84fb0e1d2aa29bc95f07a5cfc4e2542957147144a43ddba41c66
b0372985b42f3222da171340a4737e0d7a62643b2894aa27c9c66a67f95455b84f3f5226d76adc6bdf9893
a3149eb0529460651df62b0afec21370a9e92eff84062b12878ff5b19fb8029f6d089d2c7fe8576ff694b
```

Simulating the internal process at the Cable Modem:

pow(usmDHKickstartMgrPublic, CM random number, DH prime)

```
$ python3 -c 'print ( hex ( pow
( 0xef26516697b54859842f6fdc933a9aeac3750a1d50472808e62f845efb604caf5e97b117b061538a05
7b1fef375da865cc0344d36f7cc6e0cc7265aa439f423d250c20518ba8459df50d710ba30a4292d43b7fbc
ee189faa12cafdafd54c4797aac2bdb5438b5f7a8a5df28b12673b95a8a17682361815fd026960bb5ab021
dc,
0x94815e16cb9ac643050278ef80825a384f946d3c050228c440700a7cfd7b4f4acc58b095d51be81ec11e
d5129434588384e771d98d327282fc9d4ff1b81930563a0d5f988952f6b166cbcd6bb692d8c1b2c320762d
2b24d5eea0a0fde537424f2fc09f8a1d65539d26472bc45b60bbc2ea03d2fa98b996c8677e0419c4fe825d
,
0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A
08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A6
37ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE65381FFFFFFFFFFFFFFFF
) ) )'
```

Shared secret at the CM, which is exactly the same as the calculated shared secret at the SNMP manager:

```
0xf4729f7ec7f55e997060e85688b72efa84fb0e1d2aa29bc95f07a5cfc4e2542957147144a43ddba41c66
b0372985b42f3222da171340a4737e0d7a62643b2894aa27c9c66a67f95455b84f3f5226d76adc6bdf9893
a3149eb0529460651df62b0afec21370a9e92eff84062b12878ff5b19fb8029f6d089d2c7fe8576ff694b
```

The derivation of the authentication and privacy keys then follows the method in RFC 2786. There are several default parameters specified in the RFC that are used for the derivation using the PBKD (password based key derivation) algorithm.

Authentication salt value: 0x98dfb5ac

Privacy salt value: 0xd310ba6

Both keys use 16 bit lengths and 500 iterations of the algorithm.

Authentication key:

```
python3 -c 'import pbkdf2; import binascii; print ( "0x" + binascii.hexlify
( pbkdf2.PBKDF2
( 0xf4729f7ec7f55e997060e85688b72efa84fb0e1d2aa29bc95f07a5cfc4e2542957147144a43ddba41c
66b0372985b42f3222da171340a4737e0d7a62643b2894aa27c9c66a67f95455b84f3f5226d76adc6bdf98
93a3149eb0529460651df62b0afec21370a9e92eff84062b12878ff5b19fb8029f6d089d2c7fe8576ff694
b.to_bytes(128, "big" ), 0x93dfb5ac.to_bytes(4, "big" ),
iterations=500 ).read(16) ).decode() ) '
```

```
0xbe0e87f5a70e2ae6ad12a513ea029c16
```

Privacy key:

```
python3 -c 'import pbkdf2; import binascii; print ( "0x" + binascii.hexlify
( pbkdf2.PBKDF2
( 0xf4729f7ec7f55e997060e85688b72efa84fb0e1d2aa29bc95f07a5cfc4e2542957147144a43ddba41c
66b0372985b42f3222da171340a4737e0d7a62643b2894aa27c9c66a67f95455b84f3f5226d76adc6bdf98
93a3149eb0529460651df62b0afec21370a9e92eff84062b12878ff5b19fb8029f6d089d2c7fe8576ff694
b.to_bytes(128, "big" ), 0xd310ba6.to_bytes(4, "big" ),
iterations=500 ).read(16) ).decode() ) '
```

```
0x50ed4b963e81ff58227497a966bacd6c
```

These two calculated keys would then be used by the SNMP manager to communicate with the cable modem SNMP agent. When the CM reboots, or the manager random number changes, the keys must be recalculated.