To: INCITS T10 Committee
From: David L. Black, EMC
Date: 16 February 2006
Document: T10/06-103r1
Subject: SSC-3: Encrypt keys for transfer to device

## --- (0) --- Changes from prior versions

Changes from r0 to r1:
- 05-446 is now at r5
- Do not remove support for Key Format 0 (key in cleartext) - instead require that encrypted keys be supported if cleartext keys are supported. Adjust text in introduction that called for replacement of Key Format 0.
- Explain use of SCSI commands for key exchange to avoid the impression that the Device Server sends something to the initiator on its own initiative. The key exchange is realized via two successive SCSI commands issued by the initiator.
- Add open issue on device server "announcement" of what to use vs. initiator reading device server capabilities and deciding what to use.
- Clarify that NIST approval of AES GCM has not yet resulted in its addition to FIPS 140-2 (that should happen in near future). This is somewhat of a red herring as FIPS approval is *not* a goal of this design, but it is still a good reason for use of AES GCM.
- Clarify that the need for a new id for the HMAC_SHA256 PRF is not a barrier to using the HMAC_SHA256 PRF.

## --- (1) --- Introduction

Document T10/05-446r5 SSC-3: Add commands to control data encryption passes encryption keys to device servers in cleartext. Providing this as the only means of passing encryption keys is not acceptable for network-oriented SCSI transports such as Fibre Channel (FCP) and iSCSI because the keys may carry great value to an eavesdropping observer in a small amount of data. The keys are far easier to obtain and remove from the victim's infrastructure than the data they protect, and put a passive observer in a position to exploit data on an encrypted tape. Forcing users to pass keys in plaintext is an unacceptable security practice for new network protocol designs in this day and age. Users may still choose to pass keys in plaintext, but the protocol needs to make the means of encrypting the keys available so that the keys can be encrypted if the situation warrants.

The T10/05-446r5 document provides the following justification for this significant security design shortcoming:

> Future enhancements to this feature may include adding a method to encrypt the data encryption key before passing it to the device server. The addition of this single feature was proving to be significantly more complex than what was already contained in the proposal. We decided to postpone discussion of this feature until a later proposal. This proposal establishes plenty of reserved fields and page codes so that it can be added without breaking everything else.

The purpose of this document is to provide a relatively simple protocol

design
that encrypts encryption keys for transfer to the device server. The design is deliberately kept simple to enable it to be incorporated without undue delay
to development of the main tape encryption commands. This requires focusing the design on a simple problem:

Prevent an eavesdropper from learning the encryption keys.

The design also protects against a related replay vulnerability by preventing an eavesdropper from replaying any command that passes encrypted keys to the device. An eavesdropper might benefit from a replay attack if she has learned the key via other means, and is capable of injecting a SCSI command, but is not capable of functioning as a full SCSI initiator.

The goal of this design is to solve these two problems (eavesdropping and replay), and only these two problems in order to quicky specify a means of passing encrypted keys that can augment T10/05-446r5's current approach of passing keys in the clear. The resulting narrow focus yields a protocol design that deliberately does not address a number of security issues that could be addressed in a subsequent, more broadly scoped design.

The following is a list of design decisions that deliberately omit complexity and security features in order to keep the design simple:

(1) No authentication or additional authorization/access control. There is no need for the initiator or device server to prove its identity. T10/05-446r5 currently allows any initiator that can send SCSI commands to the device server to issue commands to set encryption keys. There is no need to change this; it is sufficient to identify the initiator via the I_T Nexus on which the key was sent. It is sufficient to treat the initiator's ability to send commands that the device server will execute as sufficient authorization to set encryption keys.
(2) No certificates. An immediate consequence of no authentication is no certificates. Experience in IETF and T11 indicates that it is *absolutely vital* to keep certificates and authentication out of this design in order to complete it in the time available.
(3) No persistence of encryption of keys. Keys are encrypted solely for communication to the device server, and immediately decrypted. It is not necessary to allow them to be encrypted significantly before they are sent, or enable them to be stored for later decryption.
(4) No pre-shared keys. This would violate items (1) and (3) above. This protocol for encrypting keys and transmitting them to the device server starts without pre-existing security credentials of any kind.
(5) No integration with Trusted Computing Group (TCG). TCG already has SCSI commands specified for their security protocols, but they have chosen not to share those protocols with T10. Time does not appear to permit convincing TCG to share those protocols, and the level of complexity of those protocols is not known to the author.
(6) No FIPS certification. Passing keys in the clear cannot possibly be FIPS certified, hence FIPS certification is clearly not a requirement for products based on T10/05-446r5. It is therefore not a requirement for this protocol.

(7) No support for copy managers. T10/05-446r5 does not support copy managers, so this protocol does not support them either.
(8) No RSA keys. Use of RSA asymmetric cryptography (public/private keys) makes it entirely too tempting to violate some of the above design decisions. This protocol is based on an different technique, a Diffie-Hellman key exchange.

In addition, it is important to avoid inventing new security protocols and mechanisms; this design is based on extensive reuse of protocols and techniques used in the well-established IP Security (IPsec) protocols standardized by the IETF. All references of the form [RFC nnnn] are to IETF Request for Comments documents that can be obtained from: http://www.ietf.org/rfc.html .

Note: As an early design document, not all the design details are spelled out; as agreement is reached on pursuit of this approach, more details will be filled in.

## --- (2) --- Design Overview

Encrypting one or more data encryption keys and passing them to the device server involves the following steps:
a) Algorithm and Parameter Selection: The initiator and device server determine which security algorithms, protocols, and parameters to use.
b) Key Exchange: The initiator and device server conduct a protocol that creates a secret shared between them. The protocol makes it computationally intractable for an eavesdropper to determine the shared secret. The device server also creates a key identifier and sends it to the initiator as part of this step.
c) Key Derivation: Based on the key exchange, one or more session key(s) are created for use in passing encrypted data encryption keys to the device server.
d) Wrap the data encryption key(s): Using an encryption cipher, encrypt the data encryption key(s) using the appropriate session key. This step also adds additional information that the device server needs to decrypt the encryption key and verify the decryption:
  - a key identifier to tell the device server which session key is being used
  - a sequence number to prevent replay,
  - an initialization vector (IV) for the encryption
  - an integrity check covering the original data encryption key(s) (before encrypting them) and the above items of additional information.
e) Transmit the key(s) to the device server.
f) Decrypt the wrapped data encryption key(s) and verify that they were decrypted correctly.

This protocol design proposes to use the following security techniques and mechanisms to accomplish these steps:
a) Algorithm and Parameter Selection: An "announcement" model is proposed, where there are a small number of possibilities for each item that can be selected and the initiator is required to support all possibilities. The device server selects the ones it wants to use and announces them to the initiator.

b) Key Exchange: This design proposes use of an unauthenticated Diffie-Hellman Key Exchange with nonces, based on the first exchange in the IKEv2 protocol, see Section 1.2 of [RFC 4306].
c) Session Key Derivation: This design proposes to reuse the IKEv2 mechanisms based on a pseudo-random-function (prf), see Section 2.13 and 2.14 of [RFC 4306]).
d) Wrap the keys: The proposed encapsulation format is ESP [RFC 4303]. The encryption cipher and integrity check are proposed to be provided by AES GCM [RFC 4106], a combined mode of operation for AES that has been selected by NIST for standardization (it will become part of FIPS 140-2), and another cipher operating mode, possibly AES CCM [RFC 3610, RFC 4309].
e) Transmit the Keys: SECURITY PROTOCOL OUT command with a new KEY FORMAT value.
f) Decrypt and Verify: The same ESP and AES GCM techniques as are used to wrap the key.

Each step and how it is performed are discussed in further detail below.

## --- (3) --- Negotiation via Announcement

Negotiation of security mechanisms is complex, particularly as a security negotiation design is usually required to resist downgrade attacks whereby an attacker tampers with the negotiation in a fashion that causes weaker security to be used than would have been used in the absence of the attacker's tampering.

This proposed design sidesteps the entire set of negotiation issues by using an announcement approach:
- For any mechanism or parameter that has alternatives, there are a small number of options. At least 2 are generally needed (in case one becomes unusable due to an unexpected cryptographic advances), but beyond that, a smaller number of options is better.
- An initiator is required to support every option for every parameter or mechanism (hence the desire to keep the number of options small).
- A device server selects the options that it wants to use and announces them to the initiator.
- The initiator then uses the announced options.

There is no retry; the device server says "use <this>" and <this> is what is used.

Each section below lists the options for each mechanism or parameter with alternatives. An id is listed for algorithm choices; these ids are taken from the IANA IKEv2 registry at:

http://www.iana.org/assignments/ikev2-parameters

Open Issue: The device server "annoucement" model for a) differs from the approach employed elsewhere in T10/05-446r5 where the initiator reads the device server's capabilities and decides what to use. Changing to that approach would still requires that initiators support all options for every item for interoperability, as a device server could choose to only support one option for each item. For simplicity, this design should avoid attempting to match IKEv2's complete proposal negotiation functionality.

## --- (4) --- Key Exchange

The proposed protocol uses an unauthenticated Diffie-Hellman key exchange with nonces, based on the first exchange in the IKEv2 protocol.

A Diffie-Hellman key exchange is based on the computational intractability of the discrete logarithm problem - when performing arithmetic modulo a sufficiently large prime (n), if one is given g^x mod n, it is computationally intractable to determine x when g is a generator of a sufficiently large group modulo that prime.  The combination of n and g are said to define a Diffie-Hellman group.

A Diffie-Hellman key exchange proceeds as follows:

```
  Initiator           Responder
  ---------           ---------
1) Generate a random value i      Generate a random value r
2) Compute Gi = g^i mod n     Compute Gr = g^r mod n
3) Send Gi to the Responder     Send Gr to the Initiator
4) Compute (Gr)^i mod n = g^ir mod n  Compute (Gi)^r mod n = g^ir mod n
```

The resulting shared secret is g^ir mod n.  An eavesdropper can observe g^i mod n and g^r mod n, but since it is computationally intractable for the eavesdropper to determine either i or r, the eavesdropper cannot effectively compute g^ir mod n.  A brute force attack is required, which can be made arbitrarily difficult via choice of sufficiently large n - the smallest n specified in this proposal is 2048 bits in size.
The field prime (n) and associated generator (g) are fixed numbers, and n is very large in practice.  These numbers are defined in [RFC 3526].

Exponentiation modulo a large prime can be an expensive computational operation, and hence there is a desire to reuse the results; if a responder reuses g^r mod n, it need only perform one exponentiation to complete the key exchange, and if a responder can determine that the initiator has reused g^i mod n, it can skip that exponentiation and reuse the previous g^ir mod n.  Needless to say, reuse of an old shared secret would be bad from a security standpoint, so the key exchange protocol proposed here includes randomly generated nonces (Ni, Nr) that are hashed with g^ir mod n to create the shared secret.

The resulting key exchange protocol looks like the following, where SPIr is a Security Parameters Index (SPI) that identifies the results of this key exchange, and PARMs is the Device server's announcement of the security parameters that are to be used (e.g., DH group, prf, encryption algorithm, key size).

```
  Initiator         Device Server
  ---------         -------------
1)            <-- g^r mod n, Nr, SPIr, PARMs
2)  g^i mod n, Ni, SPIi, PARMs -->
```

This key exchange is realized via the issuance of a SECURITY PROTOCOL

IN command to page 12h (step 1) followed by the issuance of a SECURITY PROTOCOL OUT command to page 12h (step 2). Specifications of page 12h that realize this protocol will need to be added to both of these commands in T10/05-446r5.

Open Issue: Is this the right way to realize the key exchange?

The initiator passes SPIi for symmetry (and to enable exact reuse of the IKEv2 algorithms), but it has no use aside from initial key generation. The initiator can set it to an arbitrary value. Both SPI values are 32 bit integers. Note that the security role of "responder" has been assigned to the Device Server in order that the SCSI initiator can be called the security "initiator", despite the fact that the device server provides the first part of the key exchange.

The nonces shall be freshly generated 128-bit truly random numbers. The parameters (PARMs) are passed in both directions to avoid any possible confusion about what is being used - they are announced by the Device Server, and returned by the initiator as verification that the initiator understood what was being announced. See Section (9) below for the specification of these parameters. The Diffie-Hellman group announced by the device server determines the size of g^i mod n and g^r mod n (as well as determining the values of g and n).

Note: Need to describe allowed reuse of DH exponentials (e.g., g^i mod n) - see Section 2.12 of [RFC 4306]. Need to describe when the participants are to generate new DH exponentials - it is possible to implement this protocol in a fashion where receipt of the initial SECURITY PROTOCOL IN command does not require the device server to generate a new DH exponential before responding.

Note: Need to provide some guidance and requirements on the number of security contexts (simultaneously valid SPIs) that a device server needs to support - in typical tape cases, the required number may be quite small.

The Diffie-Hellman exchange requires selection of a Diffie-Hellman group. The two groups for this protocol are:
- The 2048-bit group defined in Section 3 of [RFC 3526], whose id is 14.
- The 3072-bit group defined in Section 4 of [RFC 3526], whose id is 15.

The 3072-bit group is considered to have strength greater than that of a 128 bit symmetric encryption key. The 2048-bit group is considered to be somewhat weaker. As indicated in Section (3) above, initiators are required to support both groups, and device servers are required to support at least one group.

## --- (5) --- Session Key Derivation

The shared secret is calculated from the key exchange using a pseudo-random function (prf) as specified in Section 2.14 of [RFC 4306]:

```
  SKEYSEED = prf(Ni | Nr, g^ir mod n )
```

where "|" concatenates its arguments as bit strings. [RFC 4306] specifies that g^ir mod n is represented as a string of octets in big endian order padded with zeros if necessary to make it the length of the modulus (n). Ni and Nr are the nonces, stripped of any headers.

Since the proposed encryption algorithm is a combined mode (one key for both encryption and the integrity check of the data), only one session key is needed:

```
  SK_cr = prf+(SKEYSEED, Ni | Nr | SPIi | SPIr)
```

See Section 2.13 of [RFC 4306] for the definition of prf+() based on prf(). SK_cr is the first m bits of prf+ where m is the number of bits needed to key the cipher in the operating mode selected (128 bits for the combined modes used in this protocol). If separate encryption (SK_er) and integrity (SK_ar) keys are needed, they are generated from the same prf+ construction, but consume more bits:

```
  (SK_ar | SK_er) = prf+(SKEYSEED, Ni | Nr | SPIr | SPIi)
```

The Session Key Derivation requires selection of a pseudo-random function. The two functions for this protocol are:
- The HMAC_SHA1 PRF defined in [RFC 2104], whose id is 2
- The AES_XCBC PRF with 128 bit key defined in [RFC 3664], whose id is 4

As indicated in Section (3) above, initiators are required to support both PRFs, and device servers are required to support at least one PRF.

Note: IETF is in the process of updating RFC 3664, which will not result in a change to the specification of the PRF algorithm for this use, but will result in a new RFC number.

Note: The AES_XCBC PRF takes a fixed length 128 bit key, requiring application
of the following text from [RFC 4306] to the initial calculation of SKEYSEED, resulting in the first 64 bits of each nonce being used:

> If the
> negotiated prf takes a fixed-length key and the lengths of Ni and Nr
> do not add up to that length, half the bits must come from Ni and
> half from Nr, taking the first bits of each.

Open Issue: In private communication a desire has been expressed to use an HMAC_SHA256 PRF defined by the application of the HMAC construction in [RFC2104] to SHA-256 - this would replace the AES_XCBC PRF, and would need a new id taken from the private use range (this is not a barrier to use of HMAC_SHA256).

--- (6) --- Key Encryption and Wrapping

The format for wrapping an encrypted key is ESP (see [RFC 4303]) with the Next Header field set to zero (see the first diagram in Section 2 of [RFC 4303]). TFC padding is not used. The Sequence Number starts at 0 and is incremented on every use. Each SPI has its own set of sequence numbers; sequence numbers shall be used in order within an SPI.

Note: See [RFC 4303] - a significant amount of text probably needs to be imported from there to specify all the fields of ESP and how to determine their values.

Note: Need to describe IV generation for the encryption algorithms. As both GCM and CCM are counter modes, using the same IV twice with the same session key results in a high security risk. [RFC 4106] probably contains some useful text on this topic.

Open Issue: May want to put some more structure in the ESP payload than just the key(s) to be encrypted to allow its use for other things in the future - 16-bit type and length fields would be fully general, even though they would duplicate the unencrypted key length field in the set data encryption page. It's not clear that this is useful, as one can also expect other cases to determine what the ESP payload is from context (as is the case here).

Key Encryption and Wrapping requires selection of algorithms for encryption, generation of the Integrity Check Value, and associated parameters (e.g., key sizes) for those algorithms. For simplicity, this proposal employs combined modes of the AES encryption ciphers; these use a single key (SK_cr in section (4) above) for both encryption and the integrity check. The combined mode cipher algorithms and associated parameters are:
- AES GCM with a 128 bit key and 16-octet integrity check value, as specified in [RFC 4106], whose id is 20.
- AES CCM with a 128 bit key and 16-octet integrity check value, as specified in [RFC 4309], whose id is 16.

As indicated in Section (3) above, initiators are required to support both algorithms, and device servers are required to support at least one algorithm.

Open Issue: Increase key sizes to 256 bits? Allow both 128 and 256?

Open Issue: GCM is an obvious choice. CCM is the only other obvious combined mode. Use of only combined modes simplifies importing text from [RFC 4303]. An alternative that shares no algorithms with AES GCM is 3DES CBC + HMAC_SHA1.

## --- (7) --- Key Transmission

The SECURITY PROTOCOL OUT command is used to the Set Data Encryption Page, as described in T10/05-446r5. Key Format 2h is specified to be ESP wrapping of an encrypted key as specified above. Key Format 0h (cleartext keys) remains in the specification, but text needs to be added to require any implementation that supports Key Format 0h or any other inband means (including vendor specific) of passing a key or keys in cleartext to the device server to also support Key Format 2h.

## --- (8) --- Decrypt and Verify

The device server shall verify that ESP-wrapped keys are received in sequence number order for each SPI. The SPI identifies the key and parameters needed to decrypt the wrapped keys and verify that they have been received successfully. If any error occurs in this process (including out of order sequence number, and integrity check failure), the command shall be terminated with CHECK CONDITION and an ASC of KEY FORMAT ERROR (need a new ASC for that).

Note: Need to take text from [RFC 4303] specifying details of the decrypt and verify process.

--- (9) --- PARMs Parameters

The key exchange described in section 4 passes a PARMs element in both directions. That element contains six 16-bit values as fixed size fields. The allowed values for identifiers are from the IANA IKEv2 registry at http://www.iana.org/assignments/ikev2-parameters.

The six 16-bit values are:
- Key exchange protocol version number. This shall be set to 1h to indicate the protocol specified in this document.
- Diffie Hellman group identifier. The allowed values are 14d [Eh] (2048-bit group) and 15d [Fh] (3072-bit group).
- Pseudo-random function identifier. The allowed values are 2h (HMAC-SHA1) and 4h (AES XCBC)
- Encryption algorithm identifier. The allowed values are 20d [14h] (AES GCM) and 16d [10h] (AES CCM). These identifiers indicate that the Integrity Check Values are 128 bits in size.
- Key length for the Encryption algorithm. This shall be set to 128d [80h].
- Integrity algorithm. This shall be set to 0h (NONE) because combined modes that incorporate an integrity check are being used.

These represent IKEv2 transform types 4 (DH), 2 (PRF), 1 (Encryption) and 3 (Integrity), plus the Key Length attribute of the Encryption Algorithm. The latter two fixed values are included for completeness. Key Length is not included for the integrity algorithm because IKEv2 does not support variable key lengths for integrity algorithms.

If an initiator receives invalid values for any of the above parameters, it shall not complete the key exchange. If a device server receives invalid values for any of the above parameters it shall terminate the command with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.