ENDL
TEXAS

Date: 30 December 2006
To: T10 Technical Committee
From: Ralph O. Weber
Subject: Security Association Model for SPC-4

Overview

A critical element of data encryption and integrity checking algorithms is an entity called an SA (Security Associ-ation) that the participating pair of endpoints represent using a pair of indices.

- In most of the security world, a SA index is know as an SPI (Security Parameters Index).
- Because of the long-standing usage of the acronym SPI in SCSI, this proposal uses SAI (Security Association Index) as the SCSI equivalent of the security-traditional SPI.

SA information (i.e., the security parameters) are never transmitted in their entirety in any of the usual SPC-4 suspects (CDBs, parameter data, etc.). A SA is represented by two sets of parameters, one stored internally at each of the two participating endpoints. This situation produces an unusual SCSI model challenge that can only be covered by some carefully crafted model text, which is the goal of this proposal.

Revision History

r0 Original revision
r1 Made changes requested by Matt Ball and David Black.
r2 Made more changes requested by Matt Ball and David Black.
r3 Made changes requested by Gerry Houlder and Bob Nixon.
r4 Made changes requested by the September 2006 CAP working group.
r5 Change KEY(n) to KEYMAT as recommended by David Black on the T10 Reflector (17 September 2006), and modify random nonce definition so that SSC-3 can reference random number generation rules in SPC-4.
r6 Made changes requested by the November 2006 SSC and CAP working groups. Also forced DS_SAI values to be unique across all I_T nexuses.

Changes made in r6 are identified with change bars.

Definition of nonce

The definition of nonce has produced substantial discussions during the development of this proposal. Coordi-nating the SPC-4 nonce definition with the one already present in OSD and OSD-2 has been tricky. For reference, the OSD definition is reproduced here with the proposed changes indicated.

3.1.23 nonce: A value that is used one and only one time ~~and thus~~ to provide uniqueness to a value (e.g., a secure cryptographic key) in whose derivation it participates or to uniquely ~~identifies~~ identify a single instance of something (e.g., a timestamp ~~an individual OSD command, or one credential~~) ~~transacted~~ exchanged between an application client~~,~~ and a device server~~, and security manager~~.

Proposed SPC-4 Changes

Most of the text shown below is new SPC-4 material shown in black. If a subclause contains old and new material colors and strikeouts are used to identify changes and notes that are not intended for inclusion in SPC-4.

2.4 NIST References

Copies of the following approved NIST standards may be obtained through the National Institute of Standards and Technology (NIST) at http://csrc.nist.gov/publications/nistpubs/index.html.

NIST SP (Special Publication) 800-38C, *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*

Copies of the following approved NIST standards may be obtained through the National Institute of Standards and Technology (NIST) at http://csrc.nist.gov/publications/fips/index.html.

FIPS 140-2, *Annex C: Approved Random Number Generators*

FIPS 180-2 with Change Notice 1 dated February 25, 2004, *Secure Hash Standard*

2.5 IETF References

Copies of the following approved IETF standards may be obtained through the Internet Engineering Task Force (IETF) at www.ietf.org.
…
RFC 2104, *HMAC: Keyed-Hashing for Message Authentication*
RFC 3566, *The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec*
RFC 3766, *Determining Strengths For Public Keys Used For Exchanging Symmetric Keys*
RFC 4086, *Randomness Requirements for Security*
RFC 4306, *Internet Key Exchange (IKEv2) Protocol*
RFC 4344, *The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol (IKE)*

{{Note: RFC 1321, *The MD5 Message-Digest Algorithm*, is already a normative reference in SPC-4.}}

3.1 Definitions

…

3.1.b hashed message authentication code (HMAC): A type of message authentication code that is calculated using a cryptographic hash function in combination with a secret key.

3.1.c key derivation function (KDF): An algorithm that is used to derive cryptographic keying material from a shared secret and other information.

3.1.f nonce: A value that is used one and only one time to provide uniqueness to a value (e.g., a secure cryptographic key) in whose derivation it participates or to uniquely identify a single instance of something (e.g., a timestamp) exchanged between an application client and a device server.

3.1.h random nonce: A nonce (see 3.1.f) that is a secure random number (see 4.6).

3.1.l SA parameters: The parameters stored by both an application client and a device server that are associated with one SA (see 3.1.m) and identified by a pair of SAIs (see 3.1.p). See 5.13.1.2.

3.1.m Security association (SA): A relationship between an application client and device server that is used to apply security functions (e.g., data integrity checking, data encryption) to data that is transferred in either direction. See 5.13.

3.1.p Security association index (SAI): A number representing the parameters for a security association as stored internally by the application client or device server. In other security models, this value is called the security parameters index (SPI). See 5.13.

3.1.s Secure hash algorithm (SHA): A secure hash algorithm (e.g., SHA-1) specified in FIPS 180-2 with Change Notice 1 dated February 25, 2004 (see 2.4).

3.1.x Secure random number: A random number that is generated in ways that protect it from security attacks. See 4.6.
…

3.2 Symbols and acronyms

…

∇	bitwise exclusive or
‖	concatenation
AES	Advanced Encryption Standard
HMAC	Hashed Message Authentication Code (see 3.1.b)
IKEv2	Internet Key Exchange version 2 (see RFC 4306)
KDF	Key Derivation Function (see 3.1.c)
SA	Security Association (see 3.1.m)
SAI	Security Association Index (see 3.1.p)
SHA-1	Secure Hash Algorithm, 160 bits (see 3.1.s)
SHA-256	Secure Hash Algorithm, 256 bits (see 3.1.s)
SHA-384	Secure Hash Algorithm, 384 bits (see 3.1.s)
SHA-512	Secure Hash Algorithm, 512 bits (see 3.1.s)

…
{{Note: Add symbols alphabetically by description. Add acronyms alphabetically by acronym.}}

4.6 Secure random numbers

Secure Random numbers should be generated as specified by RFC 4086 (e.g., see FIPS 140-2 Annex C: Approved Random Number Generators).

If the same random number source is used generate random numbers for multiple purposes (e.g., nonces and secret keys), interactions between the two shall not be allowed to compromise secrecy. If the value sequence generated by the common random number source is predictable to any degree, then the random number values that are transmitted outside the SCSI device may provide information about the random number values that the SCSI device maintains internally, based on the reasonable assumption that an adversary knows the order in which the random numbers are obtained from the common random number source. SCSI devices shall eliminate sources of such predictability.

Compliance with RFC 4086 is one method for achieving the required independence between random number values.

…

5.13 Security Features

{{Note: All text from here to 5.13.4 is new}}

5.13.1 Security associations

5.13.1.1 Principals of security associations

Before an application client and device server begin applying security functions (e.g., data integrity checking, data encryption) to messages (i.e., data that is transferred in either direction between them), they perform a security protocol to create at least one SA (see 5.13.1.3). The output of the SA creation protocol is two sets of SA parameters (see 5.13.1.2), one that is stored by the application client and one that is stored by the device server.

In this model, SAs decouple the process of creating a security relationship from its usage in processing security functions. This decoupling allows either the creation or the usage of an SA to be upgraded in response to changing security threats without requiring both processes to be upgraded concurrently.

Figure x1 shows the relationship between application clients and device servers with respect to SAs.
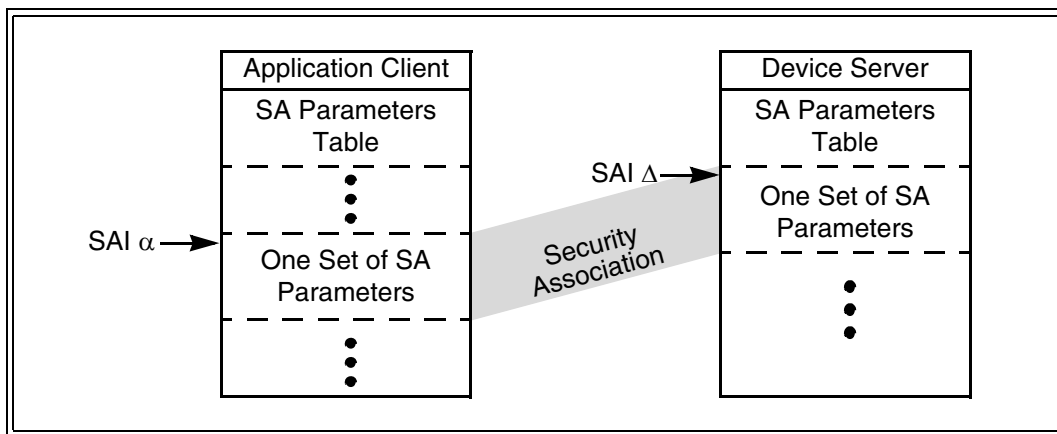

Figure x1 — SA relationships

In both the application client and the device server, the SA parameters are modelled as being stored in an indexed array and the SAI identifies one set of SA parameters within that array. The application client and device server are not required to store the parameters for any given SA in the same array locations. In order to support this implementation flexibility, a single SA is modelled as having two different SAI values (i.e., one for the application client and one for the device server).

The device server shall maintain a single SA parameters table for all I_T nexuses.

SAs shall not be preserved across a power cycle, hard reset, or logical unit reset. SAs shall not be affected by an I_T nexus loss.

5.13.1.2 SA parameters

Each SAI shall identify at least the SA parameters defined in table x1. Individual security protocols define how the SA parameters are generated and/or used by that security protocol.

Table x1 — Minimum SA parameters (part 1 of 2)

Name	Description	Size (bytes) [a]		Scope [b]
		Min.	Max.	
SA parameters that identify and manage the SA.				
AC_SAI	The SAI used by the application client to identify the SA. [c]	4	4	Public
DS_SAI	The SAI used by the device server to identify the SA. [c]	4	4	Public
TIMEOUT	The number of seconds that may elapse after the completion of an SA access operation (i.e., SA creation or SA usage by a command) before the device server should discard the state associated with this SA (e.g., the SA parameters). If SA state is discarded because no SA access operations are received during the specified interval, the device server shall respond to further attempts to access the SA as if the SA had never been created. This parameter shall not be set to zero.	4	4	Public
SA parameters that are incorporated in messages to prevent message replay attacks.				
AC_SQN	A sequence number that is incremented for each application client message on which a security function is performed.	4	8	Public
DS_SQN	A sequence number that is incremented for each device server message on which a security function is performed.	4	8	Public

[a] These size values are guidelines. Specific security protocols may place more exacting size requirements on SA parameters.
[b] Public SA parameters may be transferred outside a SCSI device unencrypted. Secret SA parameters shall be encrypted whenever they are transferred outside a SCSI device.
[c] SAI values between 0 and 255, inclusive, are reserved.
[d] Nonce SA parameters shall be at least half the size of KEY_SEED SA parameter.
[e] The number of bits of entropy in the KEY_SEED should be as close to the number of bits in the KEY_SEED as possible (see RFC 3766).

Table x1 — Minimum SA parameters (part 2 of 2)

Name	Description	Size (bytes) [a]		Scope [b]
		Min.	Max.	
SA parameters that are used by security functions to derive the secret keys that are applied to messages (e.g., for encryption).				
AC_NONCE [d]	A random nonce (see 3.1.h) value that is generated by the application client and used as an input to the key derivation security algorithm specified by the KDF_ID SA parameter during the derivation of an encryption key.	16	64	Public
DS_NONCE [d]	A random nonce value that is generated by the device server and used as an input to the key derivation security algorithm specified by the KDF_ID SA parameter during the derivation of an encryption key.	16	64	Public
KEY_SEED [e]	A value that is known only to the application client and device server that are participating in this SA that in combination with the applicable nonce is used to derive the KEYMAT value.	16	64	Secret
KDF_ID	A security algorithm (see 5.13.2) coded value that identifies the KDF used by the application client and device server.	4	4	Public
SA parameters that are used by security functions to secure messages between the application client and device server.				
KEYMAT	A value that is known only to the application client and device server that are participating in this SA that may be subdivided into one or more key values that are used in security functions that secure messages.	14	1 024	Secret

[a] These size values are guidelines. Specific security protocols may place more exacting size requirements on SA parameters.
[b] Public SA parameters may be transferred outside a SCSI device unencrypted. Secret SA parameters shall be encrypted whenever they are transferred outside a SCSI device.
[c] SAI values between 0 and 255, inclusive, are reserved.
[d] Nonce SA parameters shall be at least half the size of KEY_SEED SA parameter.
[e] The number of bits of entropy in the KEY_SEED should be as close to the number of bits in the KEY_SEED as possible (see RFC 3766).

5.13.1.3 Creating a security association

The SECURITY PROTOCOL IN command (see 6.27) and SECURITY PROTOCOL OUT command (see 6.28) security protocols shown in table x2 are used to create SAs. The process of creating an SA establishes the SA parameter (see 5.13.1.2) values as follows:

    a) Initial values for:
        A) Both (i.e., application client and device server) sequence numbers; and
        B) All KEYMAT bytes set to zero;
        and
    b) Unchanging values for:
        A) Both SAIs;
        B) Both nonces;
        C) KEY_SEED; and
        D) KDF_ID SA.

Table x2 — Security protocols that create SAs

Security Protocol Code	Description	Reference
TBD	TBD	TBD

5.13.2 Key derivation functions

5.13.2.1 Overview

Table x3 summarizes the key derivation functions defined by this standard.

Table x3 — Key derivation functions summary

Security Algorithm Code (see table x44)	Description	Reference
0002 0001h	IKEv2 iterated HMAC KDF based on MD5	5.13.2.2
0002 0002h	IKEv2 iterated HMAC KDF based on SHA-1	5.13.2.2
0002 FF01h	IKEv2 iterated HMAC KDF based on SHA-256	5.13.2.2
0002 FF02h	IKEv2 iterated HMAC KDF based on SHA-384	5.13.2.2
0002 FF03h	IKEv2 iterated HMAC KDF based on SHA-512	5.13.2.2
0002 0004h	IKEv2 iterated HMAC KDF based on AES-128 in CBC mode	5.13.2.2

Note: The Key Size column has been removed from the above table.

5.13.2.2 IKEv2 iterated application of a key derivation functions

The IKEv2-based iterative technique (see RFC 4306) for generating specified numbers of KEYMAT bits is based on any of several underlying HMAC functions and bit string sized that are particular to those HMAC functions (see table x4). The technique requires the following inputs from or related to the SA parameters:

   a) AC_SAI;
   b) DC_SAI;
   c) AC_NONCE;
   d) DC_NONCE;
   e) KDF_ID;
   f) KEY_SEED; and
   g) The number of KEYMAT bits that are to be produced.

The KDF_ID SA parameter selects the following IKEv2 iterative function parameters as shown in table x4:

   a) HFUNC (i.e., the HMAC function upon which the IKEv2 iterative process is based);
   b) MIN (i.e., the minimum number of KEY_SEED bits on which the IKEv2 iterative process is allowed to oper-
      ate);
   c) KYB (i.e., the preferred number of KEY_SEED bits); and
   d) HFB (i.e., the number of bits produced by each invocation of the HFUNC HMAC function).

Table x4 — IKEv2-based iterative key derivation functions summary (part 1 of 2)

| KDF_ID (see table x3) | IKEv2 iterative function parameters selected by KDF_ID | | | | |
	HFUNC	MIN (bits)	KYB (bits)	HFB (bits)	HMAC function description
0002 0001h	MD5	128	512	128	HFUNC is MD5 (i.e., the secure hash function defined in RFC 1321) and the single parameter to the MD5 function is the one that is identified as Message in RFC 1321.
0002 0002h	SHA-1	160	512	160	HFUNC is the SHA-1 secure hash function defined in FIPS 180-2 with Change Notice 1 (see 2.4) and the single parameter to the SHA-1 function is identified as Message in FIPS 180-2 with Change Notice 1.
0002 FF01h	SHA-256	256	512	256	HFUNC is the SHA-256 secure hash function defined in FIPS 180-2 with Change Notice 1 (see 2.4) and the single parameter to the SHA-256 function is identified as Message in FIPS 180-2 with Change Notice 1.
0002 FF02h	SHA-384	384	1 024	384	HFUNC is the SHA-384 secure hash function defined in FIPS 180-2 with Change Notice 1 (see 2.4) and the single parameter to the SHA-384 function is identified as Message in FIPS 180-2 with Change Notice 1.
0002 FF03h	SHA-512	512	1 024	512	HFUNC is the SHA-512 secure hash function defined in FIPS 180-2 with Change Notice 1 (see 2.4) and the single parameter to the SHA-512 function is identified as Message in FIPS 180-2 with Change Notice 1.

Table x4 — IKEv2-based iterative key derivation functions summary (part 2 of 2)

KDF_ID (see table x3)	IKEv2 iterative function parameters selected by KDF_ID				
	HFUNC	MIN (bits)	KYB (bits)	HFB (bits)	HMAC function description
0002 0004h	AES-128 in CBC mode	128	128	128	HFUNC is the AES-XCBC-PRF-128 secure hash function defined in RFC 4434 (see 2.5) and RFC 3566, and the single parameter to the AES-XCBC-PRF-128 function is identified as M or message in RFC 3566.

If the KEY_SEED SA parameter contains fewer than MIN bits (see table x4), the request to produce KEYMAT shall be terminated with an error. If the KEY_SEED SA parameter contains at least MIN bits, the request to produce KEYMAT shall be processed as described in this subclause.

Copy the KEY_SEED SA parameter temporary memory named SKEY, and process SKEY as follows:

1) If the number of bits copied to SKEY is greater than KYB, HFUNC shall be used to reduce SKEY to HFB bits as follows: SKEY = HFUNC(SKEY); and
2) If the number of bits in SKEY is less than KYB, SKEY shall be extended to KYB bits by appending as many zero bits as are needed to make the SKEY length equal to KYB bits.

In temporary memory named STRING store the concatenated contents of the following SA parameters:
1) AC_NONCE;
2) DC_NONCE;
3) AC_SAI; and
4) DC_SAI;

The function to be iterated is IFUNC(P1, P2) and shall be computed using HFUNC as follows:

$$\text{IFUNC}(P1, P2) = \text{HFUNC}(P1 \nabla \text{OPAD} \,\|\, \text{HFUNC}(P1 \nabla \text{IPAD} \,\|\, P2))$$

Where:

OPAD   is byte string that contains 64 bytes each of which is set to 5Ch (see RFC 2104 as identified in 2.5)
IPAD   is byte string that contains 64 bytes each of which is set to 36h (see RFC 2104 as identified in 2.5)

In principle, IFUNC is applied to STRING using SKEY to produce the KEYMAT bits. In equation notation, the operation is as follows:

$$\text{KEYMAT} = \text{IFUNC}(\text{SKEY}, \text{STRING})$$

However, accomplishing this may require multiple applications of IFUNC because the number of bits output by IFUNC is not sufficient to meet the number of KEYMAT bits that are to be produced.

The IKEv2-based iterative technique for applying IFUNC is as follows:

1) Initialize PREV_OUTPUT to a null string (i.e., a string that contains no bits);
2) Repeat the following function for values of N that increment from one by one to a maximum of 255 or until the total number of bits returned by all invocations of IFUNC equals or exceeds the number of KEYMAT bits that are to be produced, which ever occurs first:

$$T_N = IFUNC(SKEY, (PREV\_OUTPUT \parallel STRING \parallel \text{a byte containing the value N}))$$

and

3) Concatenate the T_N values (e.g., $T_1 \parallel T_2 \parallel T_3$) and return as many of the resulting bits as specified by the number of KEYMAT bits that are to be produced input parameter, starting with the first bit in T_1.

{{Note: The following subclause already appears in SPC-4, and modifications are shown.}}

5.13.1 Security algorithm codes

5.13.3 Security algorithm codes

Table x44 lists the security algorithm codes used in security protocol parameter data.

Table 44 — Security algorithm codes

Code	Description	Reference
Encryption Algorithms		
0001 0010h [a]	AES-CCM with a 16 byte MAC	NIST SP 800-38C
0001 0014h [a]	AES-GCM with a 16 byte MAC	NIST SP 800-38D
KDF Algorithms		
0002 0001h [a]	Concatenation KDF based on MD5	5.13.2.2
0002 0002h [a]	Concatenation KDF based on SHA-1	5.13.2.2
0002 0004h [a]	Concatenation KDF based on AES-128 in CBC mode	5.13.2.2
0002 FF01h [a]	Concatenation KDF based on SHA-256	5.13.2.2
0002 FF02h [a]	Concatenation KDF based on SHA-384	5.13.2.2
0002 FF03h [a]	Concatenation KDF based on SHA-512	5.13.2.2
Other Algorithms		
0000 0400h - 0000 FFFFh	Vendor specific	
All other values	Reserved	
[a] The lower order 16 bits of this code value are assigned to match an IANA assigned value, if any, for an equivalent IKEv2 encryption algorithm (see 3.1.52) and the high order 16 bits match the IANA assigned IKEv2 transform type (i.e., 1 – Encryption Algorithms, 2 – Pseudo-random Functions).		