07-069r1

Capability based Command Security

SCSI commands standard proposal

IBM Research Lab in Haifa

February 2007

Table of Contents

1 General

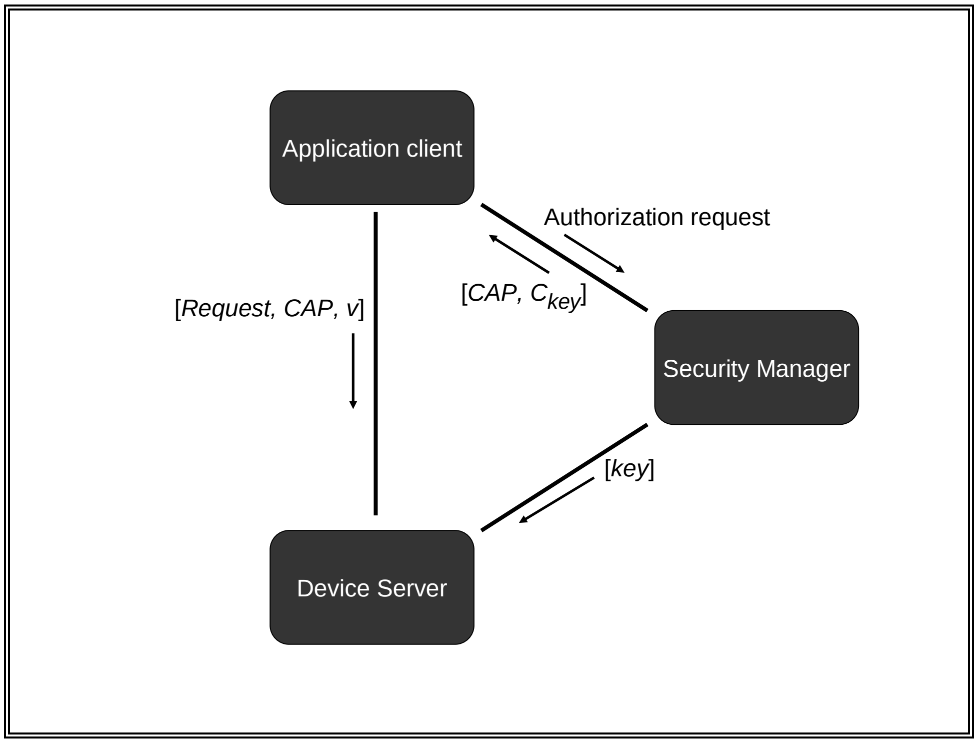
[This reset of this section is to be deleted; I think section 2 now covers all the necessary details of the CbCS model for the standard (ST).]

1.1 Overview

The Capability based Command Security (CbCS) protocol is a capability-based SCSI protocol which cryptographically enforces the integrity of the credential and its legitimate use by the client. It is a proposed extension to the SCSI Primary Commands (SPC) standard. The protocol is based on the OSD security protocol [ObsSec05] [OSD04], mapping the object access control and security protocol to logical units of any device type, with appropriate adjustments.

CbCS is a credential-based access control protocol composed of three active entities: an application client, a device server, and a security manager (see Figure 1). As a capability-based access control system, all requests to the device server must be accompanied by a capability, which encodes a set of rights the holder has on a logical unit, and be cryptographically secured.

Figure 1 - CbCS architecture



There are two steps required by the client to access a logical unit via SCSI commands:

- Prior to sending an I/O command to a SCSI target device, the client requests a credential with certain permissions from the security manager (over a secure channel) and in return the security manager sends back a pair [*CAP*, C_{key}]. Together, the capability *CAP* and capability key C_{key} form the credential. C_{key} is secret whereas *CAP* is public.

- The client sends to the SCSI target device, together with the request, the capability *CAP* along with a validation tag *v*, namely [*Request, CAP, v*]. *v* is computed by the client using C_{key}.

In order to ensure the authenticity and privacy of the commands and the data, the application client has to communicate with both the security manager and the device server over a secure channel.

As will be described below, C_{key} can be computed from *CAP* by the device server; hence the validation tag is also computable by the device server. Based upon the security method, the device server validates the validation tag and checks whether the operation requested by the command is indeed permissible by the capability *CAP*. Note that the device server does not need to authenticate the client or to have a notion of client identity.

Throughout the protocol there is a need to use a pseudo-random function as a cryptographic primitive. A field in capability specifies which function is used to compute the validation tag for the command.

The security manager and the device server share a set of symmetric secret keys that are updated periodically. A credential is based on one of the keys. When a key is updated, all credentials based on that key are no longer valid.

The details of the protocol are described and explained inside the following section, which is intended to be incorporated in SPC-4.

1.2 The Security Manager

The security manager is responsible for (1) key management (both storing and rotating the keys) across all of the target devices managed by the security manager and (2) composing appropriate credentials for a client according to a given policy.

To enable flexibility in the upper software layers, the precise security manager policies are not defined by this protocol. The standard specifies the format of the credential, which is an entity exchanged between the client, the security manager and the target device; however, the format of the policies and the methods used for determining and generating credentials based on those policies are internal to the security manager and are left out of the scope of the standard.[1]

Issues such as security method, and the type of communications channel between client and security manager are determined by the upper-layer software. This allows the upper-layer software to make decisions based on environment (e.g., secured SAN vs. insecure LAN) and usage models (e.g., sensitive corporate data vs. open web server data).

A protocol for setting and refreshing the keys and controlling security attributes on the target devices is standardized in by means of SCSI commands. For this purpose the security manager acts as a target device. It is possible to pass those commands and associated data transfers as payload of data inside any type of communications channel between the security manager and the target device, and handle its completion and status via that channel.[2]

Likewise, a protocol for an application client to authenticate and retrieve credentials from the security manager is standardized in by means of SCSI commands. Again, it is possible to pass those commands and associated data transfers as payload of data inside any type of

[1] The security manager may communicate with a separate policy manager for access decision making.
[2] Using non-SCSI mechanism to exchange the keys may be useful to save having the SCSI stack and/or FC connectivity in the security manager.

communications channel between the security manager and the target device, and handle its completion and status via that channel.[3]

To ensure correctness of the security protocol itself, the interactions between the security manager and the target device are defined using the capability model. Device servers require the security manager to present a valid capability authorizing the operation. Because security-level commands such as changing keys require a high degree of security, the security manager must use the appropriate method of security as specified for the device server with which it is interacting. This prevents the potential security weakness of attempting to set a key using a command with the NOSEC security method.

Finally, the capability includes an expiration time that limits the time a compromised capability can cause damage and helps upper-level distributed software by limiting the time a client can access a logical unit. Because the security manager encodes the expiration time within the capability and the device server interprets the expiration time, we require some degree of clock synchronization between the device server and Security manager. The protocol for synchronizing the clocks is not specified as part of the protocol, but expects that a standard clock synchronization protocol is implemented in a secure manner. Unauthorized setting of the target device's clock could cause denial of service or re-enabling of previously revoked access.

1.3 Special Considerations

Credential Revocation: Rapid revocation of access rights is a very important operation for distributed systems. Reasons for revocation could be security breach detection and SAN configuration changes. The protocol recognizes this basic function and enables immediate revocation via two mechanisms for invalidating a credential.

The first mechanism, key exchange, is a coarse-grained approach that exchanges the key between the security manager and the device server. By changing the shared key, all previous credentials a security manager had generated for the logical units of a particular device are now invalid.[4]

The second mechanism is fine-grained and invalidates all outstanding credentials for a given logical unit by utilizing the LU policy access tag. This tag is a settable LU attribute (typically by the security manager only, since it requires a special permission to set). A valid credential must match the policy access tag of the logical unit; hence, we can invalidate all outstanding credentials for a logical unit by modifying the value of its policy access tag. In order to avoid re-validation of revoked credential, policy access tags should not be reused within the lifetime of a credential.

Bootstrapping: Since a credential includes the policy access tag, which is stored as an attribute for the logical unit, we may have a problem of bootstrapping, particularly if the security manager does not have this information in its memory. How does the security manager generate a credential to read this attribute if it does not know this attribute?

To address this, the security manager has the option to generate a capability with a wild card (zero) for the policy access tag. In this case, when calculating the capability arguments, the device server should not take into account the actual value of the policy access tag associated with the logical unit. This essentially creates a credential which cannot be invalidated during its life-span other than by a key exchange.

[3] Using non-SCSI mechanism to exchange the credentials may be useful to save having the SCSI stack and/or FC connectivity in the security manager.

[4] This is assuming the same set of keys s used for all the logical units. There is also an option to have a separate set of keys for every logical unit.

Similar issue exists for the capability expiration time field, with similar solution. The device server shall skip validation of the expiration time if tat field is set to zero in the capability.

Delegation: To delegate a credential to another client, a client must transfer both *CAP* and C_{key} in a private manner (e.g., over an encrypted channel). If desired, such delegation may be prevented by use of the `Audit` field. As shown in [HKN05], an appropriate use of the `Audit` tag leads to "security confinement" thereby preventing leakage of information to unauthorized users.

Migration: The protocol has to take in account gradual migration of SANs to using access controls to logical units. First, host support and storage systems support are likely to be available at different times from different vendors. Security attributes should be settable and retrievable at the LU level. Specifically:

- A target port should be able to support both regular and secure LUs at the same time.
- An application client should be able to determine the security access controls applied to any particular LU.
- Compatibility should be maintained for old application clients such that rejecting their access to secure LUs is done in a way that allows for graceful failures. Ideally the secure LUs would not look like usable LUs to back-level application clients. Repeated errors are too much of an impact for host systems.

Changes to SPC-4

Proposal 07-029 defines ESC (Encapsulated SCSI Command) CDB format. This proposal is dependent on 07-029r1.

2 Capability based Command Security

2.1 Overview

Capability based Command Security (CbCS) is a credential-based access control system. The security model is composed of the following components:

a) A SCSI target device;

b) A security manager consisting of two SCSI devices:

   a. A security target device

   b. A security initiator device

c) Application clients.

(Editors note: This needs a UML class diagram to accurately describe the interaction of the security classes to the classes we already have. The UML should also include the secure application client and a secure device server.)

The function of the security device server is to prepare credentials in response to an application client request. A credential is a data structure containing a capability that protected by an integrity check value. The credential has the following properties:

a) Capability: Grants defined access to a SCSI logical unit for specific command functions; and

b) Integrity check value: Protects the capability and commands that include the capability from various attacks (see 2.6).

(Editors note: The following need to added to the glossary: credential, capability, integrity check value, command function, policy, and secure SCSI initiator device. Then those definitions may be removed from these sections)

Command function is one unit of work within a single command. Commands that are not bi-direction are single command functions. Bi-directional commands include multiple command functions (i.e., commands that involve both reading and writing of data).

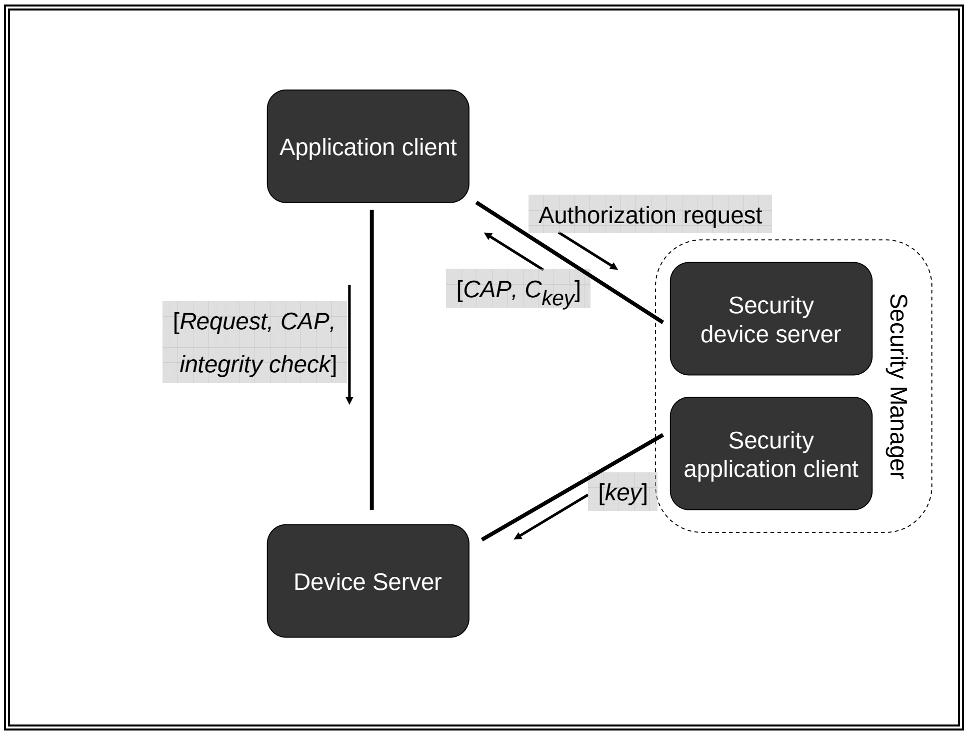
The secure application client manages access control policy. Capabilities are prepared by the secure device server based on that access control policy.

Controlling access to a logical unit requires coordination between key and security attributes set by the security application client and credentials generated by the security device server. The mechanism for coordination between the security device server and the security application client is not defined in this standard.

[Editor's note: In the OSD security model this functionality is performed by a separate policy manager entity which receives requests for capabilities from the security manager. In this description the security and policy management are both contained in a single entity called security manager. This difference does not reflect a difference in the model itself, merely in its representation.]

Figure 2 shows the flow of transactions between the components of a CbCS capable SCSI domain.

Figure 2 - The CbCS security model



2.2 Secure device server

2.2.1 Secure device server overview

The application client requests capabilities and capability keys from the secure device server A secure device server returns a capability key (C_{key}) with each credential giving the application client access to a specific logical unit. The application client sends the capability keys to that logical units device server as part of a CbCS encapsulated command which allows the device server to authenticate the capability with an integrity check value (see 2.7.2, 2.7.3).

The secure device server may authenticate the application client, but the device server does not authenticate the application client. It is sufficient for the device server to verify the capabilities and integrity check values sent by the application client.

2.2.2 Credentials

The XXXX command is used to request a credential from the secure device server (see x.x.x).

If NOSEC is not used, the device server shall validate each command received from an application client to confirm that:

a) The credential has not been tampered with (i.e., that the credential was generated by the secure device server and includes an integrity check value using a secret key known only to the secure manager and device server);

b) The credential was obtained by the application client from the secure device server or through delegation by another application client (i.e., that the application client knows the capability key that is associated with the credential and has used the capability key to provide a proper integrity check value or values for the command); and

c) The requested SCSI command is permitted by the capability in the credential (see 2.9).

2.2.3 Capabilies

The capability key allows the device server to validate a credential and determine if the capability has been tampered with (e.g., an application client that has just the capability but not the capability key is unable to generate SCSI commands with a valid integrity check value which denies access to the logical unit). Delegation of a credential is permitted, if an application client delegates both the credential and the capability key.

2.3 Secure application client

The capability keys are computed using secret keys that are shared between the secure application client and the device server. The secret keys are managed by the secure application client in conjunction with the secure SCSI initiator device. The command integrity check values are computed using capability keys. This standard includes SCSI commands for the secure application client to set and manage the secret keys stored in the target device (see 6.1, 7.1). (Editors note: Is this really the target device? Or should it be the logical unit?)

2.4 Trust assumptions

After the target device is a trusted class (i.e., after an application client authenticates that it is communicating with a specific target device or a logical unit within ithat target device using methods inside or outside the scope of this standard), the application client trusts the device server to: (Editors note: This opens this up to allowing secure access all logical units with essentially one key, it that really what is wanted?)

a) Provide integrity for stored data;

b) Perform the security protocol and functions defined for it by this standard; and

c) Not be controlled in a way that operates to the detriment of the application client's interests.

The secure device server and the secure application client are trusted classes. After the secure device server is authenticated by the application client and the secure application client is authenticated by the target device using methods inside or outside the scope of this standard, the secure device server and application client are trusted to:

a) Safely store long-lived capability keys and secret keys;
b) Apply access controls correctly according to requirements that are outside the scope of this standard;
c) Perform the security functions defined for it by this standard; and
d) Not be controlled in a way that operates to the detriment of the application client's or logical unit's interests.

(Editor's note: The above list has problems)

The application client is not a trusted class. However, the CbCS security model is defined so that the application client receives service from the device server only if it interacts with both the security device server and the device server in ways that assure the propriety of the application client's actions.

The application client is not a trusted class. However, the CbCS security is defined so that a device server only accepts SCSI command from an application client if the application client interacts with the secure device server and the device server as defined in x.x.x.

Secure application clients and secure device servers are trusted to protect capability keys from disclosure to unauthorized entities.

The secure device server, the secure application client and the device server shall maintain synchronization between their clocks.

Communications between the secure application clients, application clients, secure device servers, and device servers are trusted based on the requirements shown in Table 1

Table 1 - Communications trust requirement

Connection	**Communication trust requirement**
application client <--> device server	message integrity [a]
application client <--> secure device server	confidential
secure application client <--> device server	confidential
secure device server <--> secure application client	confidential
security devices <--> policy manager (if any)	message integrity (Ed note: This row should be deleted)
Confidential communications are protected from eavesdropping by methods outside the scope of this standard. Message integrity assures that the message received is the one that was sent (i.e., no tampering occurred). Messages in which tampering is detected shall be discarded. [a] Message integrity is required for security of the CbCS access control. Confidentiality of the data transferred between the application client and the device server may be required and	

implemented by other means. (Ed note: This sentence does not compute)

2.5 Policy Management

Policy management shall be performed by the secure application client and the secure device server:

a) The secure device server provides access policy controls to application clients using policy-coordinated capabilities; and

b) The secure application client, in concert with the target device, prevents unsecured access to a logical unit.

The policy management is confined to the secure application client and secure device server. Certain implementations may offload it to a separate policy manager entity with communications between the security manager and the policy manager to request and return credentials. (Such communication would be done in a vendor specific manner.) This variation does not affect this standard, and the policy management is described in this model as a security manager function.

(Editor note: I believe the above should be changed to: The communication of policy management information may occurs in a manner outside the scope of this standard.)

2.5.1 Capabilities

2.5.1.1 Introduction

All CbCS encapsulated commands shall contain a capability (see 2.5.1.2) that specifies the functions (e.g., read, write, attributes setting, attributes retrieval) that the device server is allowed to process in response to the encapsulated SCSI CDB.

The device server shall validate that the requested functions are allowed by the capability based on:

a) The type of functions; and

b) The logical unit.

The policies that determine which capabilities are provided to which application clients are outside the scope of this standard.

The secure device server shall deliver capabilities to application clients as follows:

a) If the security method in use for the logical unit is NOSEC (see 2.6.3), then the securie device server may:

    A) Allow application clients to prepare their own capabilities; or

    B) Coordinate the preparation of capabilities for multiple application clients in response to requests as if CAPKEY was the selected security method;

    or

b) If a security method in use for the logical unit is CAPKEY (see 2.6.4), then the secure device server shall prepare of capabilities by:

    A) Requiring application clients to request credentials and capabilities (Editor note: I don't see how this is enforceable by this standard); and

    B) Preparing capabilities only in response to application client requests.

2.5.1.2 Capability format

(Editor note: This belongs in the command section not the model section)

A capability (see Table 2) is included in a credential returned by the security device server in response to the RECEIVE CREDENTIAL command (see 8), and in a CbCS encapsulation parameters (see 4) to enable the device server to verify that the sender is allowed to perform the command functions described by the encapsulated CDB.

Table 2 - Capability descriptor format

Byte \ Bit	7	6	5	4	3	2	1	0
0	CAPABILITY FORMAT (1h)				KEY VERSION			
1	SECURITY METHOD							
2	(MSB)							
	INTEGRITY CHECK VALUE ALGORITHM							
5								(LSB)
6	(MSB)							
	CAPABILITY EXPIRATION TIME							
11								(LSB)
12								
	AUDIT							
31								
32								
	PERMISSIONS BIT MASK descriptor							
35								
36	(MSB)							
	POLICY ACCESS TAG							
39								(LSB)
40	Reserved				LU DESCRIPTOR TYPE			
41	LU DESCRIPTOR LENGTH							
42								
	LU DESCRIPTOR							
57								

The CAPABILITY FORMAT field (see Table 3) specifies the format of the capability. The capability format may also be the credential format. This standard only supports the CAPABILITY FORMAT field being set to 1h (i.e., the format defined by this standard).

Table 3 – Capability format field

Code	Description
0h	No capability
1h	The format defined by this standard

2h - Fh	Reserved

The command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB, if the CAPABILITY FORMAT field contains any value other than 1h.

The KEY VERSION field is specifies which secret key, from the set of working keys, that is being used to compute the integrity check value.

The SECURITY METHOD field should be set a valid CbCS security method (see Table 4). Valid CbCS security methods are specified in 2.6.

Table 4 specifies the CbCS security methods. CbCS security methods shall be used to specify the CbCS security methods to be used by SCSI command specified in this CbCS encapsulated command.

Table 4 – The CbCS Security methods

Code	Security Method	Description
0000h	NOSEC	No security (see 2.6.3)
0001h	CAPKEY	Integrity of capabilities (see 2.6.4)
0002h – 0FFFh	Reserved	
1000h – FFFEh	Vendor specific	
FFFFh	Reserved	

The INTEGRITY CHECK VALUE ALGORITHM field specifies the algorithm used to compute the integrity check value for this capability (see 2.7). It shall be set to a value defined in table K4 [06-449r1 - Table K4 – Integrity algorithm identifiers]

The CAPABILITY EXPIRATION TIME field specifies expiration time of the capability. The time is the number of milliseconds that have elapsed since midnight, 1 January 1970 UT. If the CAPABILITY EXPIRATION TIME field is non-zero and is less than the current time set in the device server when processing the command, the command shall be terminated with a CHECK CONDITION status, the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

If the CAPABILITY EXPIRATION TIME field contains zero, the capability has no expiration time.

The method for synchronizing the clocks is outside the scope of the CbCS model.

The AUDIT field contains a vendor specific value.

The PERMISSIONS BIT MASK descriptor (see Table 5) specifies the permissions allowed by this capability. More than one permissions bit may be set. The device server shall verify that the bits applicable to the encapsulated command are all set to one in the PERMISSIONS BIT MASK descriptor before performing the encapsulated SCSI command. In Table 5 byte 0 and byte 1 specify the permissions for all SCSI commands. In Table 5 byte 2 and byte 3 may be used by a device type specific command set standard to specify permissions unique to the device type. However, other command set standards shall not override the definition of Table 5 byte 0 and byte 1 as defined in this standard. The associations between the permissions specified in the permissions bit mask descriptor and SCSI commands are specified in the Controlled Commands CbCS page returned from the SECURITY PROTOCOL IN command specifying the CbCS security protocol (see 6.1.7).

Table 5 – PERMISSIONS BIT MASK descriptor

Byte \ Bit	7	6	5	4	3	2	1	0
0	DATA READ	DATA WRITE	ATTR READ	ATTR WRITE	SEC MGMT	Reserved		
1	Reserved							
2	For device type specific use							
3								

A DATA READ bit set to zero indicates the encapsulated SCSI command has no read permission. A DATA READ bit set to one indicates the encapsulated SCSI command has read permission.

A DATA WRITE bit set to zero indicates the encapsulated SCSI command has no write permission. A DATA WRITE bit set to one indicates the encapsulated SCSI command has write permission.

A attribute read (ATTR READ) bit set to zero indicates the encapsulated SCSI command has no attribute read permission. A ATTR READ bit set to one indicates the encapsulated SCSI command has attribute read permission.

A attribute write (ATTR WRITE) bit set to zero indicates the encapsulated SCSI command has no attribute write permission. A ATTR WRITE bit set to one indicates the encapsulated SCSI command has attribute write permission.

A security management (SEC MGMT) bit set to zero indicates the encapsulated SCSI command has no security management permission. A SEC MGMT bit set to one indicates the encapsulated SCSI command has security management permission.

If the POLICY ACCESS TAG field contains a value other than zero, the policy access tag attribute of the logical unit (see 6.1.5) is compared to the POLICY ACCESS TAG field contents as part of verifying the capability. If the POLICY ACCESS TAG field contains zero, then no comparison is made to the policy access tag attribute of the logical unit. The security application client changes the policy access tag to prevent unsafe or temporarily undesirable accesses to a logical unit (see 2.5.1.3).

If the non-zero value in the CDB POLICY ACCESS TAG field is not identical to the value in the policy access tag attribute of the logical unit (see 6.1.5), then the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

The LU DESCRIPTOR TYPE field (see Table 6) specifies the format of information in the LU DESCRIPTOR field.

Table 6 - LU DESCRIPTOR TYPE field

Code	Description
0h	Vendor specific
1h – 2h	Reserved
3h	NAA (see 7.6.3)
4h – Fh	Reserved

In order to implement credential based on logical unit unique identifier, the same identifier type shall be 1) used in the access control policies set in the policy manager; 2) returned by the device server in Device Identification VPD page (Inquiry page 83h); 3) used by the application client to identify the device and request the corresponding credential from the security device server; 4) returned by the security device server in response in the capability in response to the application client's credential request.

(Ed note: the above marked paragraph needs to stay in the model)

The LU DESCRIPTOR LENGTH field indicates the length in bytes of the LU DESCRIPTOR field. If the value of this field is greater than 16, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

2.5.1.3 Policy access tags

A secure application client may block capability-based access to a logical unit by:

a. changeing the policy access tag attribute associated with a logical unit (see 7.1.2); or

b. Changing or invalidating the secret keys shared with the device server.

2.5.1.4 Capability validation

The device server shall validate the Capability descriptor included in the CbCS encapsulation (see 4) as follows:

1) Verify that the CAPABILITY FORMAT field value is 1h. If the CAPABILITY FORMAT field value is other than 1h, then the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.
2) If the CAPABILITY EXPIRATION TIME field contains a non-zero value, then compare the CAPABILITY EXPIRATION TIME field to the current time (i.e., the current number of millisecond passed since midnight, 1 January 1970 UT). If the CAPABILITY EXPIRATION TIME field value is smaller than the current time value, then the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.
3) Verify that the LU DESCRIPTION field in the CDB Capability, according to the LU DESCRIPTOR TYPE field and LU DESCRIPTOR LENGTH field, matches this logical unit. If they don't match, then the command shall be terminated with a

CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

4) If the POLICY ACCESS TAG field in the CDB Capability descriptor contains a non-zero value, then compare the POLICY ACCESS TAG field to the Policy Access Tag of this logical unit. If they don't match, then the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.
5) Verify that the encapsulated command is permitted by the PERMISSIONS BIT MASK field in the Capability descriptor in the CDB. If the requested command is not permitted, then the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

GP stopped here

2.6 Security Methods

2.6.1 Overview

This standard defines two security methods.

- **NOSEC (No Security)**

  The device server verifies the capability allows the operation but does not verify the authenticity of the capability prior to performing an operation.

- **CAPKEY (Integrity of Capability)**

  Access Control Security is based on the protocol presented and analyzed in [ACF+02]. CAPKEY provides a mechanism to prevent an application client from forging or otherwise modifying a credential or replaying a credential over a different authenticated channel. In addition, it verifies that the application client rightfully obtained the credential it is presenting.

The security methods are summarized in Table 7. All methods use the same basic flow and the same credential structure.

Table 7 - CbCS security methods

Method	**Description**	**Without a secure channel**	**With a secure channel**
NOSEC	No security	No security	Network-level integrity
CAPKEY	Access control	End-to-end verification of credentials	+ Protection from network attacks

2.6.2 General

When the device server receives a command for a logical unit for which the CbCS bit is set to one in the standard Inquiry data (see 5) and the following applies:

a) the opcode field is set to 7Eh (i.e. Encapsulated SCSI Command CDB); and

b) the command includes encapsulation type 10h (i.e. CbCS encapsulation),

then the credential shall be validated before any other field in the CDB is validated.

When the device server receives a command for a logical unit for which the CbCS bit is set to one in the standard Inquiry data (see 5) and the following applies:

a) the opcode is other than 7Eh (i.e. not Encapsulated SCSI Command CDB); and

b) the received command requires authorization as described in 2.9,

then the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

When the device server receives a command for a logical unit for which the CbCS bit is set to one in the standard Inquiry data (see 5) and the following applies:

a) the opcode is 7Eh (i.e. Encapsulated SCSI Command CDB);

b) the command does not include encapsulation type 10h (i.e. CbCS encapsulation); and

c) the encapsulated command requires authorization as described in 2.9,

then the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

A command prepared for the CAPKEY security method may complete without errors reported by the device server if the NOSEC security method is in use.

2.6.3 The NOSEC security method

The NOSEC security method validates that the capability authorizes the encapsulated command for each CDB.

The NOSEC security method does not validate the integrity of the capability.

Preparing credentials for the NOSEC security method does not require the knowledge of any secret information and may be done by the application client without coordination with the security device server. The Capability descriptor part of the credential (see 2.5.1.2) is set with the SECURITY METHOD filed set to NOSEC. The integrity check value is set to zero.

The device server validates the capability as described in 2.5.1.4.

2.6.4 The CAPKEY security method

2.6.4.1 Overview

The CAPKEY security method validates the integrity of the capability information in each command. It provides comprehensive security when the service delivery subsystem between the device server and application client is secured via methods specified in the applicable SCSI transport protocol.

The integrity of the capability shall be validated before any other command processing actions are undertaken (i.e., before verifying that the command function requested in the CDB is permitted by the capability).

Given a credential and a channel, the protocol ties the credential to the channel via a **validation tag.** The validation tag is computed by the client as $F_{C_{key}}^{pr}(ChannelID)$,

Where:

> $ChannelID$ identifies the communication channel and is unique to this combination of initiator port, target port, and the particular I_T nexus on which they communicate.
>
> C_{key} is the capability key associated with the command (see 2.6.4.2).

$ChannelID$ is chosen in a vender specific manner by the target device. An application client may request the value of the $ChannelID$ (see 6.1.5). The device server compares the channel on which a request was received and its $ChannelID$, and verifies that the validation tag attached to the request equals $F_{C_{key}}^{pr}(ChannelID)$. The same validation tag may be used with all requests far a given credential.

The capability key (C_{key}) with which the validation tag $F_{C_{key}}^{pr}(ChannelID)$ is computed depends on the capability, ensuring the request is authenticated by the application client who obtained the credential.

As long as the same channel is used:

- The application client may reuse the capability and C_{key} on multiple commands for the same logical unit(s);
- The application client is not required to recalculate the validation tag on each command;
- The application client is required to recalculate the validation tag once per credential;
- The C_{key} and $F_{C_{key}}^{pr}(ChannelID)$ shall be calculated the first time the device server receives a given capability on a given channel and may be reused in processing every command received on the channel; and
- The device server is not required to recalculate C_{key} and $F_{C_{key}}^{pr}(ChannelID)$ each time an I_T_L nexus is established.

2.6.4.2 Computing the capability key

When preparing credentials (see 2.7) and validating credentials (see 2.7.3), the capability key shall be computed by the security device server using:

1) Algorithm: The algorithm specified in the INTEGRITY CHECK VALUE ALGORITHM field in the Capability descriptor;
2) Key: If the value of the KEY VERSION field in the Capability descriptor is nonzero, the working key specified by the KEY VERSION field, otherwise the authentication master key as the key;
3) Input data: The Capability descriptor.

The capability key is placed in the INTEGRITY CHECK VALUE field of the credential returned from the RECEIVE CREDENTIAL command.

2.6.4.3 Computing the validation tag

When preparing credentials (see 2.7.2), the validation tag shall be computed by the application client using:

1) Algorithm: The algorithm specified in the INTEGRITY CHECK VALUE ALGORITHM field in the Capability descriptor;
2) Key: The capability key (see 2.6.4.2) returned from the RECEIVE CREDENTIAL command in the INTEGRITY CHECK VALUE field;
3) Input data: The Security Token returned by the device server in the CbCS Attributes page (see 6.1.5).

The validation tag is placed in the INTEGRITY CHECK VALUE field of the credential passed by the application client in the CbCS encapsulation (see 4).

2.7 Credentials

A credential authorizes specific access to a specific logical unit. It consists of two parts: A capability and integrity check value (see Table 8). The capability descriptor (see 2.5.1) identifies the logical unit and specifies the specific access rights, as well as parameters specifying how it should be validated by the device server. The integrity check value authenticates the capability and is used for validation.

Table 8 - Credential format

Bit / Byte	7	6	5	4	3	2	1	0
0 … 57	Capability Descriptor							
58 … 121	INTEGRITY CHECK VALUE							

[Editor's note: We use this size (64 bytes) for the INTEGRITY CHECK VALUE field to accommodate the HMAC-SHA-512 algorithm. However, other algorithms consume less space (HMAC-SHA-1 consumes 20 bytes). We would like to have variable length parameter but the ESC proposal (07-029r1) does not support variable length encapsulation parameters.]

There are two types of credentials used in the CbCS protocol:

1) A credential is transferred from the security device server to an application client over a communications mechanism that meets the requirements specified in 2.4. The credential is returned in response to the RECEIVE CREDENTIAL command (see 8).
2) A credential is transferred from the application client to the device server over a communications mechanism that meets the requirements specified in 2.4. The credential is placed in the encapsulation parameters of the CDB CbCS encapsulation (see 4).

2.7.1 Preparing credentials by the secure device server

In response to a request from an application client, the secure device server shall prepare and return a credential (see Table 26) as follows:

1) If the access controls policy does not authorize the application client's request, an error shall be returned to the requesting application client;
2) Prepare the capability and insert it in the credential:
    a. Set the SECURITY METHOD field to the value of the corresponding field in the Attributes CbCS page (see 6.1.5) of the logical unit for which the credential is requested;
    b. Set the KEY VERSION field to the number of the working key secret key used to compute the credential integrity check value;
    c. Set the INTEGRITY CHECK VALUE ALGORITHM field to the value that specifies the algorithm used to compute all integrity check values related to this credential. The algorithm shall be one of those identified by the supported integrity check value algorithm attributes in the CbCS capabilities page (see 6.1.4) of the logical unit for which the credential is requested;
    d. Set the CAPABILITY EXPIRATION TIME field to a value that is consistent with the policy;
    e. Optionally set the AUDIT field in a vendor specific manner;
    f. Set the PERMISSIONS BIT MASK descriptor to a value that is consistent with the policy;
    g. Set the POLICY ACCESS TAG field to a value that matches the POLICY ACCESS TAG attribute in the Attributes CbCS page (see 6.1.5) of the logical unit for which the credential is requested. The value zero may also be set. That would produce a credential that cannot be revoked by changing the policy access tag attribute of the logical unit;
    h. Set the LU DESCRIPTOR TYPE, LU DESCRIPTOR LENGTH and LU DESCRIPTOR fields to those of the logical unit for which the credential is requested; and
    i. If the security method in use is CAPKEY, compute the capability key as described in 2.6.4.2, and place it in the INTEGRITY CHECK VALUE field in the credential. If the security method in use is NOSEC, set the INTEGRITY CHECK VALUE field to zero.
3) Return the credential thus constructed to the application client with the integrity check value serving as the capability key.

Successful use of the capability expiration time (see item d in step 2) requires some degree of synchronization between the clocks of the device server, the security application client and the security device server. The protocol for synchronizing the clocks is not specified by this model, however, the protocol should be implemented in a secure manner (e.g., it should not be possible for an adversary to set the clock in the device server backwards to enable the reuse of expired credentials). The REPORT TIMESTAMP and SET TIMESTAMP commands encapsulated with CbCS encapsulation may be used by the security application client for this purpose.

Security management commands issued by the security application client to the device server require that the integrity check value is computed using the authentication master key rather than a working key. The list of commands requiring use of the master key is in 2.8.2. If the master key is used to compute the credential integrity check value then set the KEY VERSION field in the Capability descriptor shall be set to zero.

For credentials returned by the security device server in response to the RECEIVE CREDENTIAL command (see 8), only working keys shall be used in computing the INTEGRITY CHECK VALUE field.

2.7.2 Preparing credentials by the application client

The client shall prepare the credential for sending it to the device server in the CDB CbCS encapsulation parameters as follows:

1) If the CAPKEY security method is in use, copy the Capability descriptor received from the security device server in response to the RECEIVE CREDENTIAL command into the Capability descriptor parameter of the CDB CbCS encapsulation. If the NOSEC security method is in use, prepare the Capability descriptor as described in 2.7.1.
2) If the CAPKEY security method is in use, compute the validation tag as described in 2.6.4.3 and place it in the INTEGRITY CHECK VALUE parameter of the CDB CbCS encapsulation.

2.7.3 Validating credentials by the device server

The device server shall validate the credential prior to performing any other validation on the encapsulated command or any operation requested by the encapsulated command.

The device server shall validate credentials as follows:

1) If the CAPKEY security method is in use and the SECURITY METHOD field in the Capability descriptor is other than CAPKEY, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.
2) If the CAPKEY security method is in use:
    a. Compute the capability key as described in 2.6.4.2;
    b. Compute the validation tag as described in 2.6.4.3;
    c. Compare the computed validation tag with the INTEGRITY CHECK VALUE field in the CbCS encapsulation parameters. If they don't match, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB; and
3) Verify the Capability descriptor as described in 2.5.1.4.

2.8 Secret Keys

2.8.1 Overview

All credentials are based on a secret key that is shared between the device server, the security application client that manages its security attributes, and the security device server that grants credentials to application clients. Keys shall be refreshed regularly.

Key management requirements are:

- The security application client should replace the target device keys in a secure manner even if the channel it has with the target device is not secure;
- The device server shall support multiple keys;
- The security application client shall contain a random source for generating keys;
- The device server is may contain a random source for generating keys; and
- A device server should not invalidate valid credentials under any conditions.

This standard defines a two-tier hierarchy of keys: A master key and a set of working keys. Each logical unit has a set of master key and working keys assigned to it; however, a single set of keys may be shared among multiple logical units within a device.

The working keys are used to generate the capability keys used by clients to access individual logical units. Working keys should be refreshed very frequently (e.g., hourly). A key refresh shall invalidate all credentials generated by that key. The device server may declare up to 16 refreshed versions of the key as valid (i.e., define multiple keys that are concurrently valid). To support this feature, the protocol defines a key version field that is incorporated in the capability indicating which key should be used in the validation process (see 2.5.1.2).

The number of active key versions used is agreed by mechanisms, outside the scope of this standard, between the device server and the security application client. When setting a new key, the security application client tags the key with a version number. The device server uses this tag to determine which key to use in validating a credential in a CbCS encapsulation (see 2.7.3).

There is a pair of two keys collectively called the master key: A generation master key is used to generate working keys. An authentication master key is used to generate credentials for commands to set and refresh keys, and modify device security attributes. The master key may be refreshed. Refreshing the master key is accomplished by a Diffie-Hellman key exchange algorithm that ensures forward secrecy of the master key. This algorithm is carried over a sequence of three commands:

1) SECURITY PROTOCOL OUT command specifying the CbCS protocol and the Set Master Key, Seed Exchange page (7.1.4);
2) SECURITY PROTOCOL IN command specifying the CbCS protocol and the Set Master Key, Seed Exchange page (6.1.6); and
3) SECURITY PROTOCOL OUT command specifying the CbCS protocol and the Set Master Key, Change Master Key page (7.1.5).

Separate set of master key and working keys may be used for each logical unit, or a single set may be used for all logical units served by a well-known security protocol logical unit's device server. The standard supports the following options:

a) A single set of master key and working keys is used by the device server through the well-known security protocol logical unit. This set of keys serves all the logical units within the target device;
b) A separate set of master key and working keys is used for each logical unit within the target device; and

c) A single set of master key and working keys is used by the device server through the well-known security protocol logical unit. In addition, a separate set of master key and working keys **may be** used for any logical unit within the target device. The global set of keys serves any logical unit that doesn't have its own set of keys.

The standard does **not** support the following features:

a) A single set of keys serving a distinct set of logical units within a target device. However, a secure application client may use the same keys for a set of logical units.

b) A global master key is used to generate logical unit specific working keys. This feature doesn't seem to give a significant feature on top of a single set of working keys.

2.8.2 Secret key hierarchy

Every credential prepared by the security device server includes an integrity check value that is computed using either a working key or the authentication master key.

The authentication master key shall be used in preparing credentials for the following commands:

1) SECURITY PROTOCOL OUT command specifying the CbCS security protocol and the Set Key page

2) SECURITY PROTOCOL OUT command specifying the CbCS security protocol and the Set Master Key, Seed Exchange page

3) SECURITY PROTOCOL IN command specifying the CbCS security protocol and the Set Master Key, Seed Exchange page

4) SECURITY PROTOCOL OUT command specifying the CbCS security protocol and the Set attributes CbCS page (see 7.1.2). [Editor's note: not sure the master key is required here.]

If the encapsulated command is SECURITY PROTOCOL OUT command specifying the CbCS security protocol and the Set Master Key, Change Master Key page, then the key to use is the next authentication master key computed after GOOD status has been returned by the SECURITY PROTOCOL IN command specifying the CbCS security protocol and the Set Master Key, Seed Exchange page.

For any other command, the key to use one of the working keys associated with the logical unit. The KEY VERSION field in the Capability descriptor shall be set accordingly.

2.8.3 Computing updated generation keys and new authentication keys

When processing the Set Key CbCS page (see 7.1.3) and the Set Master Key CbCS pages (see 7.1.4, 6.1.6, and 7.1.5), the device server shall compute new generation key and authentication key as follows:

The inputs generation key and authentication key are as follows:

a) The input key value is one of the following:

A) For a Set Key CbCS page (see 7.1.3), the master key generation key; or

B) For a master key computed following the processing of the Set Master Key, Change Master Key CbCS page (see 7.1.5), the previous master key generation key shall be used;

b) The seed value is one of the following; and

A) For a Set Key CbCS page, the contents of the SEED field of the Set Key CbCS page; or

B) For a Set Master Key, Change Master Key CbCS page, the value computed after successful completion of the SECURITY PROTOCOL IN command specifying the Set Master Key, Seed Exchange CbCS page (see 6.1.6) and updated by the Set Master Key, Change Master Key CbCS page (see 7.1.5).

The updated generation key shall be computed by performing the integrity check algorithm as specified in the INTEGRITY CHECK VALUE ALGORITHM field in the capability in the pertinent page using the following inputs:

a) Input key value; and

b) Seed value.

[Editor's note: Should we define a separate field encoding this algorithm rather than using the one used in the credential?]

The new authentication key shall be computed by performing the integrity check algorithm as specified in the INTEGRITY CHECK VALUE ALGORITHM field in the capability in the pertinent page using the following inputs:

a) Input key value; and

b) Seed value with the least significant bit changed as follows:

A) If the seed value least significant bit is zero, then the bit shall be changed to one; or

B) If the seed value least significant bit is one, then the bit shall be changed to zero.

2.9 CbCS interactions with commands and task management functions

2.9.1 Association between commands and command functions

The PERMISSIONS BIT MASK descriptor in the capability (see 2.5.1.2) specifies which command functions are allowed by this capability. When processing CbCS encapsulation commands, the device server shall verify that the bits applicable to the encapsulated SCSI command are all set to one in the PERMISSIONS BIT MASK descriptor before performing the request specified by the encapsulated SCSI command.

In general, the associations between commands and required permissions as specified in the PERMISSIONS BIT MASK descriptor are not specified by this standard, with the exceptions described in 2.9.2 that are essential for correct functioning of the CbCS security model. Other command set standards may specify required associations between commands and permissions pertaining to the specific device type.

The associations between commands and permissions in the device server may be queried by an application client using the SECURITY PROTOCOL IN command specifying the CbCS protocol and the Controlled Command CbCS page (see 6.1.7).

[Editor's note: Should we define a command for setting the associations reported by REPORT CBCS CONTROLLED OPERATION CODES?]

When the device server receives a command which is reported by the Controlled Command CbCS page (see 6.1.7), and the command is not encapsulated with CbCS encapsulation (see 4), the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

When the device server receives a command which is not reported by the REPORT CBCS CONTROLLED OPERATION CODES command as requiring authorization, and the command is encapsulated with CbCS encapsulation (see 4), the command may complete without errors reported by the device server.

2.9.2 Specific SPC-4 command authorization requirements

The REPORT LUNS and INQUIRY commands shall not require CbCS encapsulation. These commands are used by an application client to discover logical units and identify them.

The REQUEST SENSE and TEST UNIT READY commands shall not require CbCS encapsulation. These commands may be used by an application client to recover from and/or diagnose errors resulting from unauthorized access attempts to a CbCS protected logical unit.

The SECURITY PROTOCOL IN and SECURITY PROTOCOL OUT commands specifying the CbCS protocol shall require CbCS encapsulation (see 4), and shall require that the SEC MGMT bit is set in the PERMISSIONS BIT MASK descriptor in the Capability descriptor (see Table 5).

2.10 Security attributes

Device servers supporting CbCS may support more than one security attribute. Security attributes may be:

a) target device based;
b) logical unit based;
c) changeable; or
d) non-changeable.

Device servers may support:

a) only target device based security attributes;
b) only logical unit based security attributes; or
c) both.

If a device server supports both target based and a logical unit based security attribute is received, then the logical unit based security attribute overrides any target based security attribute on that device server.

Global attributes are queried and modified through the SECURITY PROTOCOL well-known logical unit (see x.x.x)

Table 9 specifies the CbCS attributes.

Table 9 - CbCS attributes

Security attribute name	Length (bytes)	Target device or logical unit specific	Application client settable
Supported security methods	n*2	Target device	No
Default security method	2	Target device	Yes
Security method	2	Target device/Logical unit	Yes
Supported integrity check value algorithms	n*2	Target device	No
Supported DH groups	n*2	Target device	No
Clock	6	Target device	No
Master key identifier	8	Target device / Logical unit	No [d]
Working key identifier	16*8	Target device / Logical unit	No [d]
LU initial policy access tag	4	Target device	Yes
Policy access tag	4	Logical unit	Yes
Security token	Not defined [c]	Logical unit [b]	No
Command authorization requirements	n*4 [e]	Logical unit	No [?]

[b] The security token returned by the device server is unique to the I_T nexus on which the security token is returned.

[c] The security token length is specific to the implementation of secure channel for the I_T nexus

[d] The key identifier is set by the application client when a new key is generated as described in 7.1.3 and 7.1.5. It is not settable by means of setting CbCS security attributes described in 7.1.2

[e] The 'n' in this place stands for the number of command for which CbCS encapsulation is required

The supported integrity check value algorithms security attribute provides a means for a device server to report the integrity check value algorithms it supports. Integrity check value algorithms are used to compute integrity check values. See 7.7.4.11.3 [06-449r1].

The supported DH group security attribute provides a means for a device server to report the DH groups it supports for the Diffie-Hellman key exchange with the application client that is performed as part of setting a new master key (see 7.1.4). See 7.7.4.11.4 [06-449r1].

The clock security attribute shall contain the current time in use by the device server represented as the count of the number of milliseconds elapsed since midnight, 1 January 1970 UT.

The LU initial policy access tag security attribute specifies the initial value for the policy access tag for a newly created logical unit. The initial value for this attribute shall be set to FFFF FFFFh (see 2.5.1.3).

The policy access tag security attribute specifies the expected non-zero contents of the POLICY ACCESS TAG field in any capability that allows access to this logical unit (see 2.5.1.3).

Command authorization requirements specify the commands for which the device server requires CbCS control by means of CbCS encapsulation (see 4) and what permission bits are required in the PERMISSIONS BIT MASK descriptor (see Table 5) for each command.

Setting and querying security attributes are used by the application client by issuing the SECURITY PROTOCOL IN command and SECURITY PROTOCOL OUT command with the CbCS security protocol (see 6.1, 7.1).

3 ENCAPSULATION TYPE definitions

Following is a proposed change to table x3 from 07-029r1.

Table x3 — OUTERMOST ENCAPSULATION TYPE field and NEXT ENCAPSULATION TYPE field

		Size of encapsulation parameter descriptors (bytes)		
Code	**Description**	**Prefix**	**Postfix**	**Reference**
00h	No next layer	n/a	n/a	4.3.4.2.1
Prefix and Postfix codes				
01h – 07h	Reserved			
Prefix only codes				
xxh	Capability based Commands Security	98	0	
xxh - FFh	Reserved		0	

4 Capability based Commands Security encapsulation

[Editor's note: I'm not sure where exactly this section should be added in SPC-4.]

The Capability based Commands Security (CbCS) encapsulation allows the application of security to a SCSI command using the parameters specified in this subclause.

Support for CbCS encapsulation type is mandatory if the CbCS bit in standard INQUIRY data (see 5) is set to one.

The encapsulated CDB may be any valid CDB (i.e., any CDB defined in any SCSI standard).

Table 10 - CbCS encapsulation descriptor format

Bit / Byte	7	6	5	4	3	2	1	0
0	NEXT ENCAPSULATION TYPE							

Bit / Byte	7	6	5	4	3	2	1	0
1	Reserved							
2 … 59	Capability descriptor							
60 … 123	INTEGRITY CHECK VALUE							

The Capability descriptor is defined in 2.5.1.2.

The Capability descriptor and the INTEGRITY CHECK VALUE field shall be prepared by the application client as described in 2.7.2.

The device server shall validate the Capability descriptor and the INTEGRITY CHECK VALUE field as described in 2.7.3.

5 Standard INQUIRY data

Change in 6.4.2 Standard INQUIRY data:

The standard INQUIRY data (see table 85) shall contain at least 36 bytes.

Table 85 — Standard INQUIRY data format

Bit / Byte	7	6	5	4	3	2	1	0
0	PERIPHERAL QUALIFIER			PERIPHERAL DEVICE TYPE				
1	RMB	Reserved						
2	VERSION							
3	Obsolete	Obsolete	NORMACA	HISUP	RESPONSE DATA FORMAT			
4	ADDITIONAL LENGTH (n-4)							
5	SCCS	ACC	TPGS		3PC	**CbCS**	Reserved	PROTECT
6	Obsolete	ENCSERV	VS	MULTIP	Obsolete	Obsolete	Obsolete	ADDR16[a]
7	Obsolete	Obsolete	WBUS16[a]	SYNC[a]	Obsolete	Obsolete	CMDQUE	VS
8	(MSB)	T10 VENDOR IDENTIFICATION						
15								(LSB)
16	(MSB)	PRODUCT IDENTIFICATION						
31								(LSB)
32	(MSB)	PRODUCT REVISION LEVEL						
35								(LSB)

Bit / Byte	7	6	5	4	3	2	1	0
36	Vendor specific							
55								
56	Reserved				CLOCKING[a]		QAS[a]	IUS[a]
57	Reserved							
58	(MSB)	VERSION DESCRIPTOR 1						
59								(LSB)
	. . .							
72	(MSB)	VERSION DESCRIPTOR 8						
73								(LSB)
74	Reserved							
95								
	Vendor specific parameters							
96	Vendor specific							
n								

[a] The meanings of these fields are specific to SPI-5 (see 6.4.3). For SCSI transport protocols other than the SCSI Parallel Interface, these fields are reserved.

A Capability based Command Security (CbCS) bit set to one indicates that the SCSI target device supports Capability based Command Security (see 2). A CbCS bit set to zero indicates that the SCSI target device does not support Capability based Command Security.

If the CbCS bit is set to one, the target device is required to support the following commands and parameters:

- Encapsulated SCSI command (see 4.3.4.2 [07-029r1]);
- CbCS encapsulation type (see 4);
- SECURITY PROTOCOL IN specifying the CbCS security protocol (see 6.1).
- SECURITY PROTOCOL OUT specifying the CbCS security protocol (see 7.1).

6 Changes in SECURITY PROTOCOL IN command

[Changes in section 6.29 SECURITY PROTOCOL IN command]

6.29.1 SECURITY PROTOCOL IN command description

Table 186 — SECURITY PROTOCOL field in SECURITY PROTOCOL IN command

Code	Description	Reference
00h	Security protocol information	6.29.2
01h - 06h	Defined by the TCG	3.1.128
07h	Capability based Command Security	6.29.3
08h - 1Fh	Reserved	
20h	Tape Data Encryption	SSC-3
21h - EDh	Reserved	
EEh	Authentication in Host Attachments of Transient Storage Devices	IEEE P1667
EFh	ATA Device Server Password Security	TBD
F0h - FFh	Vendor Specific	

6.1 CbCS SECURITY PROTOCOL

(New section in SPC-4: 6.29.3)

6.1.1 Overview

(New section in SPC-4: 6.29.3.1)

The SECURITY PROTOCOL IN command specifies the CbCS protocol requests the device server to return the security attributes of the:

a) logical unit; or

b) target device that contains the addressed well-known SECURITY PROTOCOL logical unit.

The command supports CbCS pages that may be requested one at a time. An application client requests a CbCS page by using a SECURITY PROTOCOL IN command with the SECURITY PROTOCOL field set to 07h (CbCS protocol) and the SECURITY PROTOCOL SPECIFIC field set to the requested CbCS page code.

The SECURITY PROTOCOL SPECIFIC field (see Table 11) specifies the CbCS pages.

Table 11 – SECURITY PROTOCOL SPECIFIC field

Code	Description	Support	Reference
0000h	SECURITY PROTOCOL IN supported CbCS page	M	6.1.2
0001h	SECURITY PROTOCOL OUT supported CbCS page	M	6.1.3
0002h-000Fh	Reserved		
0010h	Capabilities CbCS page	M/O (?)	6.1.4
0011h	Attributes CbCS page	M	6.1.5

Code	Description	Support	Reference
0012h	Set Master Key, Seed Exchange CbCS page	M	6.1.6
0013h	Controlled Commands CbCS page	M	6.1.7
0014h – FFFFh	Reserved		
Support key: M – Mandatory for device servers that support the CbCS. O – Optional			

6.1.2 SECURITY PROTOCOL IN supported CbCS page

(New section in SPC-4: 6.29.3.2)

Table 12 specifies the format of the SECURITY PROTOCOL IN supported CbCS page.

Table 12 - SECURITY PROTOCOL IN supported CbCS page

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)	PAGE CODE (0000h)						
1								(LSB)
2	(MSB)	PAGE LENGTH (n-3)						
3								(LSB)
4	(MSB)	SECURITY PROTOCOL IN supported CbCS page (first)						
5								(LSB)
	⋮							
n-1	(MSB)	SECURITY PROTOCOL IN supported CbCS page (last)						
n								(LSB)

The SECURITY PROTOCOL IN supported CbCS page shall contain a list of all of the CbCS pages the device server supports for the SECURITY PROTOCOL IN command specifying the CbCS protocol in ascending order beginning with page code 0000h.

6.1.3 SECURITY PROTOCOL OUT supported CbCS page

(New section in SPC-4: 6.29.3.3)

Table 13 specifies the format of the SECURITY PROTOCOL IN supported CbCS page.

Table 13 - SECURITY PROTOCOL IN supported CbCS page format

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)	PAGE CODE (0001h)						

Byte \ Bit	7	6	5	4	3	2	1	0
1								(LSB)
2	(MSB)	PAGE LENGTH (n-3)						
3								(LSB)
4	(MSB)	SECURITY PROTOCOL IN supported CbCS page (first)						
5								(LSB)
	. . .							
n-1	(MSB)	SECURITY PROTOCOL IN supported CbCS page (last)						
n								(LSB)

The SECURITY PROTOCOL IN supported CbCS page shall contain a list of all of the CbCS pages that the device server supports for the SECURITY PROTOCOL OUT command specifying the CbCS protocol in ascending order.

6.1.4 Capabilities CbCS page

(New section in SPC-4: 6.29.3.4)

Table 14 specifies the format of the Capabilities CbCS page.

Table 14 – Capabilities CbCS page format

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)	PAGE CODE (0010h)						
1								(LSB)
2	(MSB)	PAGE LENGTH (i*2+j*2+k*2+8)						
3								(LSB)
4	GKS	LUKS	GSMS	LUSMS	Reserved			
5	Reserved							
6		Number of supported security methods (i)						
7								(LSB)
8		SUPPORTED SECURITY METHOD (first)						
9								(LSB)
	. . .							
i*2+6	SUPPORTED SECURITY METHOD (last)							

Byte \ Bit	7	6	5	4	3	2	1	0
i*2+7								(LSB)
i*2+8	Number of supported integrity check value algorithms (j)							
i*2+9								(LSB)
i*2+10	SUPPORTED INTEGRITY CHECK VALUE ALGORITHM (first)							
i*2+11								(LSB)
	. . .							
i*2+j*2+8	SUPPORTED INTEGRITY CHECK VALUE ALGORITHM (last)							
i*2+j*2+9								(LSB)
i*2+j*2+10	Number of supported D-H groups (k)							
i*2+j*2+11								(LSB)
i*2+j*2+12	SUPPORTED D-H GROUP (first)							
i*2+j*2+13								(LSB)
	. . .							
i*2+j*2+k*2+10	SUPPORTED D-H GROUP (last)							
i*2+j*2+k*2+11								(LSB)

A Global Keys Support (GKS) bit set to one specifies that the device server supports global keys. A Global Keys Support (GKS) bit set to zero specifies that the device server does not support global keys.

A Logical Unit Keys Support (LUKS) bit set to one specifies that the device server supports per-logical unit keys. A Logical Unit Keys Support (LUKS) bit set to zero specifies that the device server does not support per-logical unit keys.

A device server that supports CbCS shall support either global keys or per-logical unit keys, and may support both. (Editor note – This requirement belongs in a model section)

A Global Security Method Support (GSMS) bit set to one specifies that the target device that contains this logical unit supports global security method (i.e., contains a SECURITY PROTOCOL well known logical unit). A Global Keys Support (GKS) bit set to zero specifies that the device server requires the security methods to be assigned to each logical unit.

A Logical Unit Security Method Support (LUSMS) bit set to one specifies that the device server supports per-logical unit security method. A Logical Unit Keys Support (LUSMS) bit set to zero specifies that the device server does not support per-logical unit security method.

A device server that supports CbCS shall support either global security method or per-logical unit security method, and it may support both. (Editor note – This requirement belongs in a model section)

The SUPPORTED SECURITY METHOD fields contain coded values of the security methods supported by the device server (see 2.6).

The SUPPORTED INTEGRITY CHECK VALUE ALGORITHM fields contain coded values of the algorithm to compute integrity check values supported by the device server (see 2.7).

The SUPPORTED DH GROUP attributes contain coded values identifying the supported values in the DH_GROUP field of Set Master Key, Seed Exchange page (see 7.1.4).

6.1.5 Attributes CbCS page

(New section in SPC-4: 6.29.3.5)

Table 15 specifies the format of the Attributes CbCS page.

Table 15 - Attributes CbCS page format

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)	PAGE CODE (0010h)						
1								(LSB)
2	(MSB)	PAGE LENGTH (n-3)						
3								(LSB)
4	(MSB)	SECURITY METHOD						
5								(LSB)
6	(MSB)	POLICY ACCESS TAG						
9								(LSB)
10		MASTER KEY IDENTIFIER						
17								(LSB)
18		WORKING KEY IDENTIFIER 0						
25								(LSB)
	⋮							
138		WORKING KEY IDENTIFIER 15						
145								(LSB)
146	(MSB)	CLOCK						
151								(LSB)
152	Reserved							

Byte \ Bit	7	6	5	4	3	2	1	0
153	SECURITY TOKEN LENGTH							
154	SECURITY TOKEN							
n								(LSB)

If the addressed logical unit is the well-known security protocol logical unit:

a) the SECURITY METHOD field is the security method assigned by the W-LUN's device manager to a new logical unit;

b) the POLICY ACCESS TAG field is the initial policy access tag assigned by the W-LUN's device server to a new logical unit; and

c) if the device server does not support global security method (i.e., the GSMS bit is set to zero in the Capabilities CbCS page), then:

   A) the SECURITY METHOD field is undefined; and

   B) the MASTER KEY IDENTIFIER field and all the WORKING KEY IDENTIFIER fields should contain FFFF FFFFh

If the addressed logical unit is not the well-known security protocol logical unit:

a) the SECURITY METHOD field is the current security method used for the addressed logical unit;

b) the POLICY ACCESS TAG field is the current policy access tag assigned to the addressed logical unit; and

c) if the device server does not support per-logical unit security method (i.e., the LUSMS bit is set to zero in the Capabilities CbCS page), then:

   A) the SECURITY METHOD field is undefined; and

   B) the MASTER KEY IDENTIFIER field and all the WORKING KEY IDENTIFIER fields should contain FFFF FFFFh.

Security methods are described in detail in 2.6.

If keys are supported for the addressed logical unit, the values of those fields are as follows:

a) The MASTER KEY IDENTIFIER field shall contain the key identifier value from the most recent successful SECURITY PROTOCOL OUT command specifying the CbCS security protocol and the Set Master Key, Change Master Key page (see 7.1.5). If that command has never been processed, then the MASTER KEY IDENTIFIER field shall contain FFFF FFFEh; and

b) Each KEY IDENTIFIER field contains the key identifier value from the most recent successful SECURITY PROTOCOL OUT command specifying the CbCS security protocol and the Set Key page, with the KEY VERSION field set to the pertinent key (0-15) (see 7.1.3). If a key is invalid (e.g., never set, invalidated by a Set Master Key, Change Master Key page, or invalidated by a Set Key page), the pertinent KEY IDENTIFIER field should contain 0000 0000h.

The CLOCK field shall contain the current time in use by the device server represented as the count of the number of milliseconds elapsed since midnight, 1 January 1970 UT.

Note: The clock field may be usable for other purposes. Perhaps it can be moved to a more generic place, e.g. mode page…

For the CAPKEY security method, the SECURITY TOKEN field contains a value that is unique to the I_T nexus on which the SECURITY PROTOCOL IN command was sent. The security token shall be random as defined by RFC 1750. An I_T nexus loss event or reset event (see SAM-4) shall cause the security token to change.

6.1.6 Set Master Key, Seed Exchange CbCS page

(New section in SPC-4: 6.29.3.6)

Table 16 specifies the format of the Set Master Key, Seed Exchange CbCS page.

Table 16 - Set Master Key, Seed Exchange CbCS page format

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)	PAGE CODE (0012h)						
1								(LSB)
2	(MSB)	PAGE LENGTH (n-3)						
3								(LSB)
4	DH DATA							
n								

If a SECURITY PROTOCOL IN command specifying Set Master Key, Seed Exchange CbCS page is received and no SECURITY PROTOCOL OUT command specifying Set Master Key, Seed Exchange CbCS page has been received on the same I_T_L nexus during the past ten seconds, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

A device server that receives a SECURITY PROTOCOL OUT command specifying Set Master Key, Seed Exchange CbCS page on one I_T_L nexus may terminate the command with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to SYSTEM RESOURCE FAILURE if any of the following command processing is incomplete on a different I_T_L nexus:

a) SECURITY PROTOCOL OUT command specifying Set Master Key, Seed Exchange CsCB page (see 7.1.4);

b) SECURITY PROTOCOL IN command specifying Set Master Key, Seed Exchange CsCB page (see 6.1.6); or

c) SECURITY PROTOCOL OUT command specifying Set Master Key, Change Master Key CsCB page (see 7.1.5).

The DH DATA field contains the device server DH_data computed as follows:

a) A random number, y, is generated having a value between zero and DH_prime minus one observing the requirements in RFC 1750; and

b) The device server DH_data is equal to DH_generator^y modulo DH_prime, where the DH_generator and DH_prime values are identified by the code value in the DH GROUP field of the most recent SECURITY PROTOCOL OUT command specifying Set Master Key, Seed Exchange CbCS page.

After GOOD status has been returned for SECURITY PROTOCOL OUT command and before the SECURITY PROTOCOL OUT command specifying the Set Master Key, Change Master Key page is processed, the next authentication master key and next generation master key shall be computed as described in 2.8.3, using a seed value that is the concatenation of the following:

1) DH_generator^{xy} modulo DH_prime; and

2) The whole content of the Device Identification VPD page (83h) returned from the addressed logical unit for the INQUIRY command (see 7.6.3).

6.1.7 Controlled Commands CbCS page

(New section in SPC-4: 6.29.3.7)

The Controlled Commands CbCS page (see Table 17) returns a list of commands for which the addressed logical unit requires CbCS control using the CbCS command encapsulation (see 4), and encapsulation requirements for the command.

Table 17 - Controlled Commands CbCS page format

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)			PAGE CODE (0012h)				
1								(LSB)
2	(MSB)			PAGE LENGTH (n*8)				
3								(LSB)
4				Command descriptor 1				
11								
				⋮				
n*8-4				Command descriptor n				
n*8+3								

Each command descriptor (see Table 18) contains information about a single supported command CDB.

Table 18 - Command descriptor format

Byte \ Bit	7	6	5	4	3	2	1	0

Byte \ Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE							
1	Reserved							
2	(MSB)	SERVICE ACTION						
3								(LSB)
4	(MSB)	PERMISSIONS BIT MASK descriptor						
7								(LSB)

The OPERATION CODE field contains the operation code of a command requiring CbCS encapsulation.

The SERVICE ACTION field contains a service action the operation code indicated by the OPERATION CODE field, which requires CbCS encapsulation. If the operation code indicated in the OPERATION CODE field does not have a service action, the SERVICE ACTION field shall be set to 00h.

The PERMISSIONS BIT MASK descriptor format is defined in Table 5. For each command function performed by the command indicated by the OPERATION CODE and the SERVICE ACTION fields, the corresponding bit shall be set to one.

[Editor's note: Should we provide a SECURITY PROTOCOLOUT page for setting permissions bit masks for a command?]

7 Changes in SECURITY PROTOCOL OUT command

[Changes in section 6.30 SECURITY PROTOCOL OUT command]

<Unchanged text here>

The SECURITY PROTOCOL field (see table 191) specifies which security protocol is being used.

Table 191 — SECURITY PROTOCOL field in SECURITY PROTOCOL OUT command

Code	Description	Reference
00h	Reserved	
01h - 06h	Defined by the TCG	3.1.128
07h	Capability based Command Security	
08h - 1Fh	Reserved	
20h	Tape Data Encryption	SSC-3
21h - EDh	Reserved	
EEh	Authentication in Host Attachments of Transient Storage Devices	IEEE P1667
EFh	ATA Device Server Password	TBD

Code	Description	Reference
	Security	
F0h - FFh	Vendor Specific	

7.1 CbCS SECURITY PROTOCOL

(New section in SPC-4: 6.30.1)

7.1.1 Overview

The SECURITY PROTOCOL OUT command specifying CbCS protocol is used to configure the CbCS attributes in the device server.

The command supports CbCS pages that may be sent one at a time. An application client requests to send a CbCS page by using a SECURITY PROTOCOL OUT command with the SECURITY PROTOCOL field set to 07h (CbCS protocol) and the SECURITY PROTOCOL SPECIFIC field set to the CbCS page code requested.

The SECURITY PROTOCOL SPECIFIC field (see Table 19) specifies the type of CbCS page that the application client is sending.

Table 19 - SECURITY PROTOCOL SPECIFIC field values

Code	Description	Support	Reference
0000h – 0010h	Reserved		
0011h	Set Attributes CbCS page	O	7.1.2
0012h	Set Key CbCS page	M	7.1.3
0013h	Set Master Key, Seed Exchange CbCS page	M	7.1.4
0014h	Set Master Key, Change Master Key CbCS page	M	7.1.5
0015h – FFFFh	Reserved		
Support key: M – Mandatory for device servers that support the CbCS protocol O – Optional			

If the SECURITY PROTOCOL SPECIFIC field is set to a reserved or unsupported value, the device server 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.

7.1.2 Set attributes CbCS page

(New section in SPC-4: 6.30.1.2)

Table 20 specifies the format of the Set Attributes CbCS page.

Table 20 – Set Attributes CbCS page format

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)	PAGE CODE (0011h)						
1								(LSB)
2	(MSB)	PAGE LENGTH (6)						
3								(LSB)
4	(MSB)	SECURITY METHOD						
5								(LSB)
6	(MSB)	POLICY ACCESS TAG						
9								(LSB)

The PAGE LENGTH field indicates the number of bytes of parameter data to follow. If the page length value results in the truncation of any field, the device server 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 PARAMETER LIST.

The SECURITY METHOD field specifies a new security method to apply to the addressed logical unit (see 2.6). The SECURITY METHOD field is set to:

a) the reserved value (FFFFh) to specify no change shall be made to the current security method;

b) a value equal to the current security method shall not be considered an error; and

c) a value that does not match any of the supported security methods reported in the Capabilities CbCS page (see 6.1.4), shall cause the device server to terminate the SECURITY PROTOCOL OUT command with CHECK CONDITION status and set the sense key to ILLEGAL REQUEST and the additional sense code to INVALID FIELD IN PARAMETER DATA.

The POLICY ACCESS TAG field specifies a new policy access tag for the addressed logical unit. The value set to 0000 0000h specifies no change shall be made to the current policy access tag value. If the addressed logical unit is the well-known SECURITY PROTOCOL logical unit and the POLICY ACCESS TAG field contains any value other than 0000 0000h, the device server shall terminate the SECURITY PROTOCOL OUT command with CHECK CONDITION status and set the sense key to ILLEGAL REQUEST and the additional sense code to INVALID FIELD IN PARAMETER DATA.

This command shall be authorized and shall be sent encapsulated by a CbCS encapsulation (see 2.9.2).

7.1.3 Set Key CbCS page

(New section in SPC-4: 6.30.1.3)

Table 21 specifies the Set Key CbCS page format.

Table 21 - Set Key CbCS page format

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)	PAGE CODE (0012h)						
1								(LSB)
2	(MSB)	PAGE LENGTH (30)						
3								(LSB)
4	Reserved							
5	Reserved				KEY VERSION			
6	(MSB)	KEY IDENTIFIER						
13								(LSB)
14	(MSB)	SEED						
33								(LSB)

The KEY VERSION field specifies the key version to be updated.

The KEY IDENTIFIER field specifies a unique identifier to be associated with the new key. The key identifier value shall be associated with the attribute specified in the Attributes CbCS page (see 6.1.5).

The SEED field contains a random number generated from a good source of entropy (e.g., as described in RFC 1750).

The updated key value shall be computed as described in 2.8.3.

This command shall be authorized and shall be sent encapsulated with CbCS encapsulation (see 2.9.2).

7.1.4 Set Master Key, Seed Exchange CbCS page

(New section in SPC-4: 6.30.1.4)

Table 22 specifies the Set Master Key, Seed Exchange CbCS page format.

Table 22 - Set Master Key CbCS page format

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)	PAGE CODE (0013h)						
1								(LSB)
2	(MSB)	PAGE LENGTH (n-3)						
3								(LSB)
4	(MSB)	DH GROUP						
5								(LSB)

Byte \ Bit	7	6	5	4	3	2	1	0
6	(MSB)	DH DATA LENGTH						
9								(LSB)
10	DH DATA							
n								

If a device server receives a SECURITY PROTOCOL OUT command specifying Set Master Key, Seed Exchange CbCS page on one I_T_L nexus and any of the following command processing is incomplete on a different I_T_L nexus:

c) SECURITY PROTOCOL OUT command specifying the CbCS protocol and the Set Master Key, Seed Exchange CbCS page (7.1.4);

d) SECURITY PROTOCOL IN command specifying the CbCS protocol and the Set Master Key, Seed Exchange CbCS page (6.1.6); or

e) SECURITY PROTOCOL OUT command specifying the CbCS protocol and the Set Master Key, Change Master Key CbCS page (7.1.5),

then the device server may terminate the command with a CHECK CONDITION status, setting the sense key to ILLEGAL REQUEST and the additional sense code to SYSTEM RESOURCE FAILURE

Editors note: the above belongs in a model section

The DH GROUP field contains the Diffie-Hellman group (see 7.7.4.11.4 [06-449r1]) that identifies the DH_generator value and DH_prime value to be used in the seed exchange. If the value in the DH GROUP field is not listed in one of the SUPPORTED D-H GROUP fields in the Capabilities CbCS page (see 6.1.4), then the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

The DH DATA LENGTH field specifies the number of bytes of the DH DATA field.

The DH_DATA field contains the DH data and is computed as follows:

DH data = DH_generator^{x} modulo DH_prime

Where:

X	value between zero and DH_prime minus one as defined in RFC 1750;
DH_generator	defined in the DH GROUP field; and
modulo DH_prime	defined in the DH GROUP field.

7.1.5 Set Master Key, Change Master Key CbCS page

(New section in SPC-4: 6.30.1.5)

Table 23 specifies the format of the Set Master Key, Change Master Key CbCS page.

Table 23 - Set Master Key, Change Master Key CbCS page format

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)	PAGE CODE (0014h)						
1								(LSB)
2	(MSB)	PAGE LENGTH (n-3)						
3								(LSB)
4	(MSB)	KEY IDENTIFIER						
11								(LSB)
6	(MSB)	APPLICATION CLIENT DATA LENGTH (k-9)						
9								(LSB)
10	APPLICATION CLIENT DH DATA							
k								
k+1	(MSB)	DEVICE SERVER DATA LENGTH (n-(k+4))						
k+4								(LSB)
k+5	DEVICE SERVER DH DATA							
n								

The KEY IDENTIFIER field specifies a unique identifier to be associated with the new master key. The key identifier value shall be placed into the MASTER KEY IDENTIFIER field in the Attributes CbCS page (see 6.1.5) before the command completes.

Table 24 specifies special key identifiers that shall not be used when setting keys. Any value not listed in the table is permitted for use by the application client when setting keys.

Table 24 - Special key identifiers

Value	Description
0000-0000h	A key that was never set or was invalidated
FFFF-FFFEh	An initial key set by the device server
FFFF-FFFFh	Pertinent key that is not supported by the device server

If the value of the KEY IDENTIFIER field contains a value that is listed in Table 24, then the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The APPLICATION CLIENT DATA LENGTH field specifies the number of bytes that follow in the APPLICATION CLIENT DH DATA field.

The APPLICATION CLIENT DH_DATA field contains the DH_data from the last SECURITY PROTOCOL OUT command specifying Set Master Key, Seed Exchange CbCS page on the same I_T_L nexus on which this command was received.

The DEVICE SERVER DATA LENGTH field contains the length in bytes of the DEVICE SERVER DH DATA field.

The DEVICE SERVER DH DATA field contains the device server DH_data from the last SECURITY PROTOCOL IN command specifying Set Master Key, Seed Exchange CbCS page on the same I_T_L nexus on which this command was received.

If the content of the APPLICATION CLIENT DATA LENGTH field of the SECURITY PROTOCOL OUT commands Set Master Key, Seed Exchange CbCS page does not match the content of the:

a) DH DATA LENGTH field in a SECURITY PROTOCOL OUT Set Master Key, Seed Exchange CbCS page that was processed on this I_T_L nexus since a I_T nexus loss event, logical unit reset event, or reset event (see SAM-4);
b) DH DATA field in a SECURITY PROTOCOL OUT Set Master Key, Seed Exchange CbCS page that was processed on this I_T_L nexus since a I_T nexus loss event, logical unit reset event, or reset event;
c) PAGE LENGTH field in a SECURITY PROTOCOL IN Set Master Key, Seed Exchange CbCS page that was processed on this I_T_L nexus since a I_T nexus loss event, logical unit reset event, or reset event; or
d) DH DATA field in a SECURITY PROTOCOL IN Set Master Key, Seed Exchange CbCS page that was processed on this I_T_L nexus since a I_T nexus loss event, logical unit reset event, or reset event,

then the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

On successful completion of a SECURITY PROTOCOL OUT command specifying the CbCS protocol and the Set Master Key, Change Master Key CbCS page shall:

a) Replace the authentication master key with the next authentication master key computed after the return of GOOD status for the most recent SECURITY PROTOCOL IN command specifying the CbCS protocol and the Set Master Key, Seed Exchange CbCS page (see 6.1.6);
b) Replace the generation master key with the next generation master key computed after the return of GOOD status for the most recent SECURITY PROTOCOL IN command specifying the CbCS protocol and the Set Master Key, Seed Exchange CbCS page; and
c) Invalidate all the working keys on the logical unit.

For every key that is invalidated by this command, the associated key identifier attribute shall have its attribute set to zero.

The next authentication master key computed after the return of GOOD status for the most recent SECURITY PROTOCOL IN command specifying the CbCS protocol and the

Set Master Key, Seed Exchange CbCS page (see 6.1.6) shall be used to compute the capability key for this command.

8 RECEIVE CREDENTIAL command

[A new sub-section in section 6 of SPC-4]

The RECEIVE CREDENTIAL command (see Table 25) provides allows the application client to receive a credential for use in the CbCS encapsulation (see 4).

Table 25 - RECEIVE CREDENTIAL CDB format

Bit / Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE (7Fh)							
1	CONTROL							
2 - 6	Reserved							
7	ADDITIONAL CDB LENGTH (n-7)							
8	(MSB)	SERVICE ACTION (1800h)						
9								(LSB)
10	(MSB)	ALLOCATION LENGTH						
11								(LSB)
12	Reserved				LU DESCRIPTOR TYPE			
13	LU DESCRIPTOR LENGTH							
14 - n	LU DESCRIPTOR							

If a RECEIVE CREDENTIAL command is received before a SECURITY PROTOCOL IN command has completed successfully on the same I_T_L nexus as the RECEIVE CREDENTIAL command was received with following field settings:

a) SECURITY PROTOCOL field set to xxh (i.e., IKEv2-SCSI); and

b) the SECURITY PROTOCOL SPECIFIC field set to 0103h (i.e., authentication phase),

then the command shall be terminated with a CHECK CONDITION status, the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

[Editor's: note: The above paragraph is based on 06-449r1 and depends on it.]

The ALLOCATION LENGTH field is defined in 4.3.4.6.

The LU DESCRIPTOR TYPE field specifies the format of information in the LU DESCRIPTOR field. If the LU DESCRIPTOR TYPE field is set to a value other than the values specified in Table 6, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

The LU DESCRIPTOR LENGTH field indicates the length in bytes of the LU DESCRIPTOR field. If the value of this field is greater than 16, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

8.1 RECEIVE CREDENTIAL parameter data

The RECEIVE CREDENTIAL parameter data is defined in Table 26.

Table 26 - Credential format

Bit / Byte	7	6	5	4	3	2	1	0
0	CREDENTIAL FORMAT (1h)				Reserved			
1	Reserved							
2	(MSB)	CREDENTIAL LENGTH (n-3)						
3								(LSB)
4	(MSB)	CAPABILITY LENGTH (k-5)						
5								(LSB)
6	Capability descriptor							
k								
k+1	(MSB)	INTEGRITY CHECK VALUE LENGTH (n-(k+4))						
k+4								(LSB)
k+5	INTEGRITY CHECK VALUE							
n								

The CREDENTIAL LENGTH field indicates the length in bytes of the capability that follow.

The CAPABILITY LENGTH specifies the length of the Capability field.

The format of the Capability is defined in 2.5.1.2.

INTEGRITY CHECK VALUE LENGTH specifies the length of the INTEGRITY CHECK VALUE field.

The INTEGRITY CHECK VALUE field contains a value that the application client shall use for preparing credentials (see 2.7.2).

9 Misc. changes

9.1 Change in C.3.5 Variable length CDB service action codes

The variable length CDB service action codes assigned by this standard are shown in table C.8.

Service Action Code	Description
1800h	RECEIVE CREDENTIAL
1801h – 1FFFh	Reserved

9.2 Change in Table 48 — Commands for all device types

In section 6.1, add the RECEIVE CREDENTIAL command to table 48.

10 Issues

- The OSD standard says that when security method other than NOSEC is used, reservation commands (including persistent reservations) shall be treated as invalid. Should this apply to CbCS? What is the impact on working systems?
- The OSD standard says:

  If the root object or any partition in the OSD logical unit is using any security method other than NOSEC, all SAM-3 task management functions except QUERY TASK shall be ignored and responded to as if they have been successfully processed. The PERFORM TASK MANAGEMENT FUNCTION command (see 6.16) allows SAM-3 task management functions to be processed under the protection of the current security method.

  Should this apply to CbCS? What is the impact on working systems?

11 References

[ObsSec05] Michael Factor, David Nagle, Dalit Naor, Erik Riedel, and Julian Satran. The OSD Security Protocol, September 2005. http://ieeeia.org/sisw/2005/PreProceedings/04.pdf

[OSD04] R. O. Weber. SCSI Object-Based Storage Device Commands – 2 (OSD-2), Date: 2004/10/04, Rev: 00. InterNational Committee for Information Technology Standards (formerly NCITS), October 2004. http://www.t10.org/drafts.htm.

[BCK96] M. Bellare, R. Canetti, and H. Krawczyk. Message authentication using hash functions: The hmac construction. RSA Laboratories' Crypto-Bytes, 2(1), Spring 1996.

[FIPS180-02] Federal Information Processing Standards Publication 180-2: SECURE HASH STANDARD.

Date: August 1, 2002.
http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

[FIPS198-02] Federal Information Processing Standards Publication 198: The Keyed-Hash Message Authentication Code (HMAC).
Date: March 6, 2002.
http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf

[HKN05] Shai Halevi, Paul A. Karger and Dalit Naor. Enforcing confinement in distributed storage and a cryptographic model for access control, 2005. Cryptology ePrint Archive: Report 2005/169.

[ACF+02] Alain Azagury, Ran Canetti, Michael Factor, Shai Halevi, Ealan Henis, Dalit Naor, Noam Rinetzky, Ohad Rodeh, and Julian Satran. A two layered approach for securing an object store network. In Proceedings of the First International IEEE Security in Storage Workshop, pages 10–23, Greenbelt, MD, 11 December 2002.