Computational Storage Architecture and Programming Model

Version 1.2

ABSTRACT: ***This SNIA document defines recommended behavior for hardware and software that supports Computational Storage.***

SNIA Standard

April 21, 2026

USAGE

DISCLAIMER

The information contained in this publication is subject to change without notice. The SNIA makes no warranty of any kind with regard to this specification, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The SNIA shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this specification.

Revision History

Major	Revision	Editor	Date	Comments
0.1	R1	Stephen Bates	02-Feb-2019	Initial conversion of the NVM Programming Model document.
0.1	R2	Stephen Bates	11-Mar-2019	Added initial draft of Scope section with some initial feedback
0.1	R3	Stephen Bates David Slik	02-April-2019	Added diagram to Scope section and introduced “Computational Function” as a term.
0.1	R4	Nick Adams	04-Apr-2019	Updates per Morning Session on 03-Apr-2019
0.1	R5	Scott Shadley	24-Apr-2019	Updates added per F2F meeting in Boston
0.1	R6	Stephen Bates and Bill Martin	21-Aug-2019	Removed resolved comments, added Theory of Operation, added illustrative diagrams, added placeholder for an illustrative example. Also performed a large amount of editorial clean-up. Added forward and removed taxonomy and abstract (approved by TWG on August 28th 2019).
0.1	R7	Stephen Bates and Bill Martin	XX Oct 2019	Update foreword section. Add references, abbreviations and definitions.
0.1	R8	David Slik	12 Nov 2019	Moved fields into tables, updated diagrams.
0.1	R9	Stephen Bates	14 Nov 2019	Minor editorial changes
0.1	R10	Bill Martin	20 Nov 2019	Incorporated approved dictionary terms, abbreviations, and reference modifications
0.1	R11	Bill Martin	11 Dec 2019	Incorporated NetInt proposal for Video Compression CSS Incorporated ScaleFlux proposal for Database Filter CSS Removed previously accepted changes
0.1	R12	Stephen Bates	15 Dec 2019	Incorporated DellEMC proposal for Hashing and CRC Removed list of CSSes at start of Section 5. Remove reference to how to add new CSSes to Section 5. Update the list of contributors as per the list on the Wiki Resolved and removed comments. Added illustrative examples from Eideticom and Arm/NGD Systems in their current form and started merging them correctly into this document.. Incorporated Seagate proposal for large dataset PCSS
0.1	R13	Stephen Bates	18 Dec 2019	Removed Annex B (Illustrative Examples)
0.3	R1	Stephen Bates	18 Dec 2019	Update author list Slight fix to Philip’s PCSS example Add Raymond’s FCSS example Minor corrections to document.
0.4	R1	Bill Martin	8 April 2020	Added PCSS on a Large Multi-Device Dataset Added PCSS – Linux Container Example
0.4	R2	Bill Martin	20 May 2020	Added FCSS – Data Deduplication Added PCSS – OPEN CL example
0.4	R3	Bill Martin	29 July 2020	Updated example in B.4 Data Deduplication Example Added Data Compression Illustrative Example Added Data Filtration Illustrative Example
0.4	R4	Bill Martin	5 August 2020	Added Scatter Gather illustrative example
0.5	R1	Bill Martin	10 Aug 2020	Minor editorial cleanup Removed editor’s notes in preparation for public review release
0.5	R2	Bill Martin	12 Aug 2020	Modifications created during TWG call on August 12. There are some partial sentences as the discussion was ended due to the end of the call.
0.5	R3	Bill Martin	8 September	Includes editor’s comments removed from public review version Includes comments received to date (Intel (Kim) and HPe (Chris)) Changes for the terminology of CSS, FCSS, PCSS, and CSP

0.5	R4	Bill Martin	30 September	Renaming of components (e.g., addition of Computational Storage Engine, Computational Storage Service becomes Computational Storage Function)
0.5	R5	Bill Martin	9 November	Changes from meetings Modified figures as requested Added memory definitions Removed resolved comments
0.5	R6	Bill Martin	10 Dec 2020	Removed change tracking R4 to R5 Fixed figure 1.1 to include CFM Added brief descriptions of CSFM, FDM, AFDM in 4.1 Moved example discover request and response to an annex
0.5	R7	Scott Shadley	16 Dec 2020	Comments from TWG meeting
0.5	R8	Scott Shadley	1 Jan 2021	Comments from TWG meeting
0.5	R9	Bill Martin	28 Jan 2021	Updated direct/indirect model Moved discovery/configuration request/response to annex Other editorial changes from Jan 27, 2021 meeting
0.5	R10	Bill Martin	28 Jan 2021	Removed redline and completed comments
0.5	R11	Bill Martin	10 Feb 2021	Resolved a number of comments from TWG meetings See individual comments with resolutions
0.5	R12	Bill Martin	11 Feb 2021	Removed redline and completed comments
0.5	R13	Bill Martin	16 Feb 2021	Modifications from TWG meeting
0.5	R14	Bill Martin	23 Mar 2021	Updated Large Data Set CSFs Updating Appendix B, sect 5 (Compression) and 6 (data filtering) illustrative examples to align with the CSx/CSE/CSF terminology. (change) Removed all resolved comments
0.5	R15	Bill Martin	23 April 2021	Updates from work group meetings with change bars
0.5	R16	Bill Martin	27 April 2021	Changes to 4.2 and 4.3 as agreed to in TWG meeting Incorporated architecture model changes agreed in TWG meeting Incorporated changes to section 5 as agreed in TWG meeting Incorporate changes for Appendix B from 4/14/2021
0.5	R17	Bill Martin	27 April 2021	Accepted changes and removed resolved comments
0.5	R18	Bill Martin	27 May 2021	Added Configuration flow chart per proposal Updated Open CL CSS per proposal Updated figures in Usage section Updated Usage definition to make consistent Moved open comments to “editor’s notes”
0.5	R19	Bill Martin	3 June 2021	Incorporated modifications to B.2
0.8	R0	Arnold Jones	9 June 2021	Formatted as a Working Draft for release outside the TWG per TWG approval ballot.
0.8	R1	Bill Martin	31 March 2022	Added initial security considerations Fixed must/shall wording throughout (3 instances) Added CSEE definition
0.8	R2	Bill Martin	1 June 2022	Removed track changes from previous revision Added modifications from SNIA-CS-Arch-Prog-Model-Theory-of-op-mods-06-01-2022.
0.8	R3	Bill Martin	21 June 2022	Removed previous change tracking Incorporated all RFC comments Resolved most RFC comments
0.8	R4	Bill Martin	22 June 2022	Removed previous change tracking Incorporated changes requested at TWG meeting All comments marked “done”

0.8	R5	Bill Martin	23 June 2022	Clean version – all comments removed
0.9		Bill Martin		Changed header from 0.8R5 to 0.9 for member vote and public review
1.0	R1	Bill Martin	12 Dec 2022	Incorporated security updates
1.0	R1a	Bill Martin	9 January 2022	Commented version of 1.0r1
1.0.1		Bill Martin	1 May 2024	Incorporation of Sequencing of commands includes resolution of reference to 5.1.1 from 1 May TWG meeting Numbering changed to be consistent with the API and general SNIA numbering policy
1.0.2		Jason Molgaard	8 May 2024	Changed CSF Completion Output (section 4.5.3) and added comments for items to correct in section 4.5
1.0.3		Jason Molgaard	28 May 2024	Added deactivation text to sections 4.3.1 and 4.3.2 to indicate that pre-activated CSEEs and CSFs may not be able to be deactivated
1.0.4		Bill Martin	10 July 2024	Accepted all previous changes in preparation for RFC ballot
1.0.5		Bill Martin	8 January 2025	All RFC comments are addressed and all change bars removed. This is to proceed to a TWG vote for forwarding to the TC for processing towards a membership vote.
1.0.6		Bill Martin	21 January 2025	Editorial changes from Solidigm
1.1.1		Jason Molgaard	07 May 2025	Incorporation of sequencing enhancements for parallel execution and passing of inputs, outputs, and parameters.
1.1.2		Jason Molgaard	11 June 2025	All RFC comments addressed from 1.1.1 and change bars removed.
1.1.3		Jason Molgaard	25 June 2025	Resolved RFC comments from 1.1.2 and removed change bars.
1.1.4		Jason Molgaard	23 July 2025	Clarifications to security assumptions for multi-tenancy.
1.2		T. Mancuso	21 April 2026	Approved SNIA Standard

Table of Contents

Table of Figures

FOREWORD

The SNIA Computational Storage Technical Working Group was formed to establish architectures and software computation in its many forms to be more tightly coupled with storage, at both the system and drive level. An architecture model and a programing model are necessary to allow vendor-neutral, interoperable implementations of this industry architecture.

This SNIA specification outlines the architectural models that are defined to be Computational Storage. As this specification is developed, requirements in interface standards and specific APIs may be proposed as separate documents and developed in the appropriate organizations.

1 Scope

This specification focuses on defining the capabilities and actions that are able to be implemented across the interface between Computational Storage devices (CSxes) (e.g., Computational Storage Processors, Computational Storage Drives, and Computational Storage Arrays) and either Host Agents or other CSxes.

The actions mentioned above are associated with several aspects of a CSx:

- **Management**: Actions that allow Host Agent(s), based on security policies, to perform:
  - **Discovery:** Mechanisms to identify and determine the capabilities and Computational Storage Resources (CSR);
  - **Configuration**: Programming parameters for initialization, operation, and/or resource allocation;
- **Security:** Considerations for security related to CSxes; and
- **Usage**: Allows a Host Agent or CSx to offload Computational Storage tasks to a CSx, including providing the target CSx with information about data locality both local to the CSx or resident on one or more non-local locations.

This specification makes no assumptions about the physical nature of the interface between the Host Agent and CSx(s). This specification and the actions associated with it will be implemented across a range of different physical interfaces. This specification also makes no assumptions about the storage protocols used by Host Agents and CSx(s).

The following storage protocols between the Host Agent and the CSx may be supported:

- **Logical Block Address**. Data is grouped into fixed-size logical units and operations are atomic at that unit size. Data is indexed via a numerical index into the Logical Block Address.
- **Key-Value**. Data is not fixed-size and is indexed by a key.
- **Persistent Memory**. Byte addressable non-volatile memory.

This specification defines actions for passing data through multiple Computational Storage Functions (CSFs) that may or may not reside on a single CSx. Additionally, it defines actions for requesting multiple Computational Storage Functions to perform a set of tasks.

2 References

The following referenced documents are indispensable for the application of this document.

SNIA Computational Storage API	Computational Storage API v1.1, available from https://www.snia.org/tech_activities/publicreview
NVMe® 2.0	NVM Express Base Specification 2.0, Approved standard, available from http://nvmexpress.org
FIPS 140-3	Federal Information Processing Standards Publication 140-3 (FIPS PUB 140-3) Security Requirements for Cryptographic Modules, March 2019, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf
ISO/IEC 19790	ISO/IEC 19790:2012(E), Information technology — Security techniques — Security requirements for cryptographic modules, August 2012, https://www.iso.org/standard/52906.html
ISO/IEC 24759	ISO/IEC 24759:2017(E), Information technology — Security techniques — Test requirements for cryptographic modules, March 2017, https://www.iso.org/standard/72515.html
ISO/IEC 15408 (multi-part standard)	ISO/IEC 15408-1:2009, Information technology — Security techniques — Evaluation criteria for IT security — Part 1: Introduction and general model, December 2009, https://www.iso.org/standard/50341.html
	ISO/IEC 15408-2:2008, Information technology — Security techniques — Evaluation criteria for IT security — Part 2: Security functional components, August 2008, https://www.iso.org/standard/46414.html
	ISO/IEC 15408-3:2008, Information technology — Security techniques — Evaluation criteria for IT security — Part 3: Security assurance components, August 2008, https://www.iso.org/standard/46413.html

3 Definitions, abbreviations, and conventions

For the purposes of this document, the following definitions and abbreviations apply.

3.1 Definitions

3.1.1 Aggregator CSF

CSF that executes an ordered sequence of CSFs

3.1.2 Allocated Function Data Memory

Function Data Memory (FDM) that is allocated for a particular instance of a CSF

3.1.3 Computational Storage

architectures that provide computation coupled to storage, offloading host processing or reducing data movement.

3.1.4 Computational Storage Array (CSA)

storage array that contains one or more CSEs.

3.1.5 Computational Storage Device (CSx)

Computational Storage Drive, Computational Storage Processor, or Computational Storage Array.

3.1.6 Computational Storage Drive (CSD)

storage element that contains one or more CSEs and persistent data storage.

3.1.7 Computational Storage Engine (CSE)

component that is able to execute one or more CSFs

Note 1 to entry Examples are: CPU, FPGA.

3.1.8 Computational Storage Engine Environment (CSEE)

operating environment for a CSE

Note 1 to entry Examples are: Operating System, Container Platform, eBPF, and FPGA Bitstream.

3.1.9 Computational Storage Function (CSF)

a set of specific operations that may be configured and executed by a CSE.

Note 1 to entry Examples are: compression, RAID, erasure coding, regular expression, encryption.

3.1.10 Computational Storage Processor (CSP)

component that contains one or more CSEs for an associated storage system without providing persistent data storage

3.1.11 Computational Storage Resource (CSR)

resources available in a CSx necessary for that CSx to perform computation

Note 1 to entry Examples are: FDM, Resource Repository, CPU, memory, FPGA resources)

3.1.12 entity

for security purposes, an actual or abstract thing that exists, did exist, or might exist, including associations among these things

Note to entry 1 CSx, CSE, CSEE, and CSF are all entities.

3.1.13 Function Data Memory (FDM)

device memory used for storing data that is used by the Computational Storage Functions (CSFs) and is composed of allocated and unallocated Function Data Memory

3.1.14 party

for security purposes, a natural person, legal person or a group of either or both, whether or not incorporated

3.1.15 platform firmware

the collection of all device firmware on a platform

3.1.16 tenant

one or more users (i.e., entities or natural persons) sharing access to a set of physical and virtual resources

3.2 Keywords

In the remainder of the specification, the following keywords are used to indicate text related to compliance:

3.2.1 mandatory

a keyword indicating an item that is required to conform to the behavior defined in this standard

3.2.2 may

a keyword that indicates flexibility of choice with no implied preference; “may” is equivalent to “may or may not”

3.2.3 may not

keywords that indicate flexibility of choice with no implied preference; “may not” is equivalent to “may or may not”

3.2.4 need not

keywords indicating a feature that is not required to be implemented; “need not” is equivalent to “is not required to”

3.2.5 optional

a keyword that describes features that are not required to be implemented by this standard; however, if any optional feature defined in this standard is implemented, then it shall be implemented as defined in this standard

3.2.6 shall

a keyword indicating a mandatory requirement; designers are required to implement all such mandatory requirements to ensure interoperability with other products that conform to this standard

3.2.7 should

a keyword indicating flexibility of choice with a strongly preferred alternative

3.3 Abbreviations

AFDM Allocated Function Data Memory

CSA Computational Storage Array

CSD Computational Storage Drive

CSE Computational Storage Engine

CSEE Computational Storage Engine Environment

CSF Computational Storage Function

CSP Computational Storage Processor

CSR Computational Storage Resources

CSx Computational Storage devices

FDM Function Data Memory

SSD Solid State Disk

4 Theory of Operation

4.1 Overview

This section describes the theory of operations for Computational Storage Devices (CSxes), Computational Storage Resources (CSRs), Computational Storage Engines (CSEs), Computational Storage Engine Environments (CSEEs), and Computational Storage Functions (CSFs).

Computational Storage architectures enable improvements in application performance and/or infrastructure efficiency through the integration of compute resources (outside of the traditional compute & memory architecture), either directly with storage or between the host and the storage. The goal of these architectures is to enable parallel computation and/or to alleviate constraints on existing compute, memory, storage, and I/O.

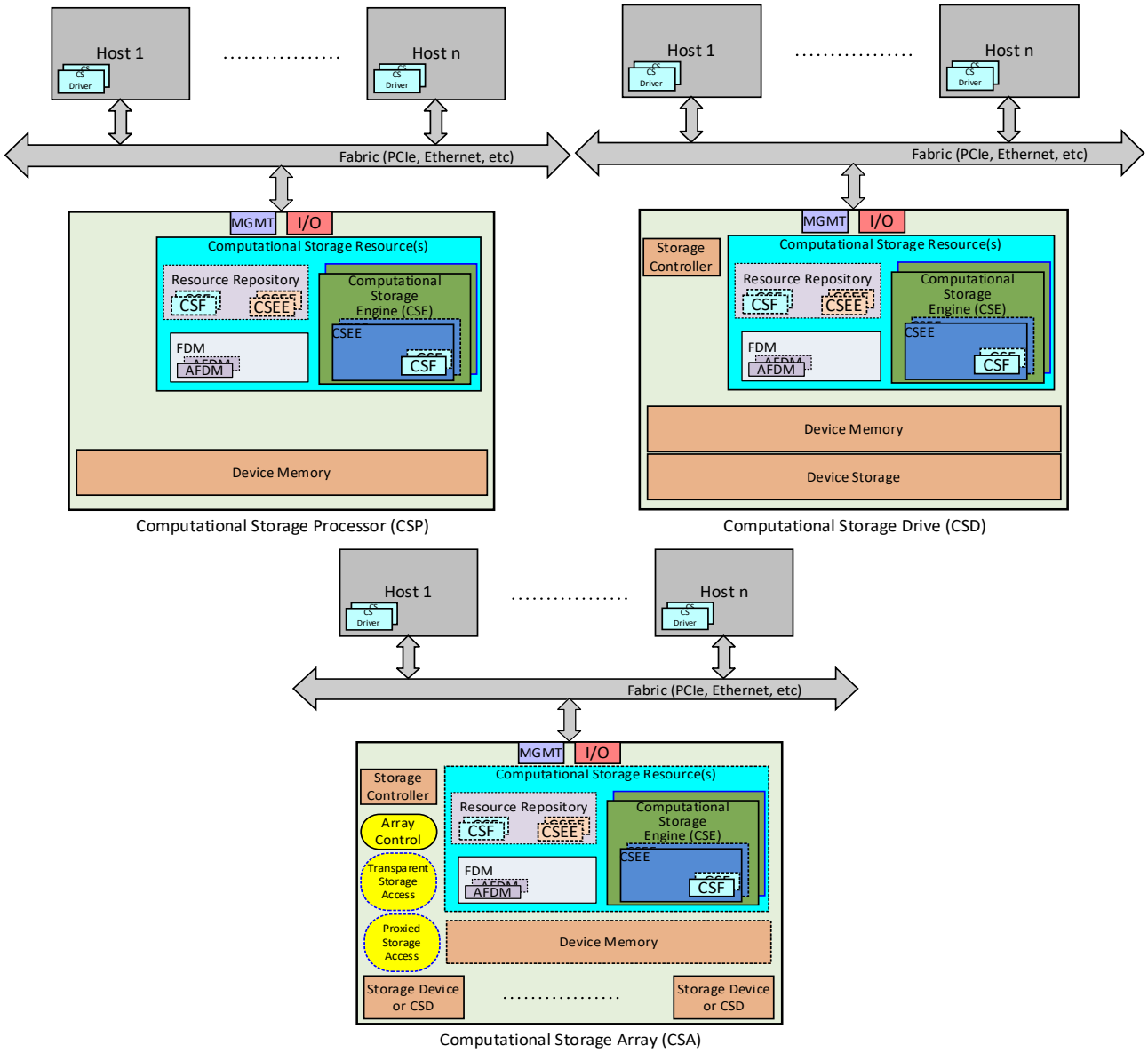

Figure 4.1– An Architectural view of Computational Storage

An illustrative example of Computational Storage devices (CSxes) is shown in Figure 4.1. A CSx consists of the following components:

- Computational Storage Resources (CSR) which contain:
  - A Resource Repository where the following may be stored:
    - Computational Storage Functions (CSFs); and
    - Computational Storage Engine Environments (CSEEs);
  - Function Data Memory (FDM) which may be used as Allocated Function Data Memory (AFDM); and
  - One or more Computational Storage Engines (CSEs);
- A Storage Controller for CSD or an Array Controller for CSA;

- Device Memory; and
- Device Storage for CSD and CSA.

Computational Storage Resources (CSRs) are the resources available in a CSx necessary for that CSx to store and execute a CSF.

A Computational Storage Engine (CSE) is a CSR that is able to be programmed to provide one or more CSFs.

A Computational Storage Engine Environment (CSEE) is an operating environment for the CSE.

A Computational Storage Function (CSF) is a set of specific operations that may be configured and may be executed by a CSE in a CSEE.

Activation is the process of associating a CSEE with a CSE, or associating a CSF with a CSEE. As part of activation of a CSEE, any resources that are necessary for that CSEE to be used on the CSE are assigned. As part of activation of a CSF, any resources that are necessary for that CSF to be used on the CSEE are assigned to the CSF. When a CSEE association with a CSE or a CSF association with a CSEE is no longer required, the CSEE or CSF may be deactivated. This deactivation process releases any assigned resources.

A CSE is required to have a CSEE activated to be able to execute a CSF in that CSEE. A CSE has FDM associated with it. A CSE is able to have one or more CSEEs with one or more CSFs activated in them at the time of manufacture that are usable by the host via management and I/O interfaces. One or more CSEEs and one or more CSFs may be downloaded by the host into the Resource Repository. These CSEEs and CSFs may then be activated. A CSE may have CSFs that have been programmed at the time of manufacture that are not changeable (i.e., not stored in the Resource Repository) (e.g., compression, RAID, erasure coding, regular expression, encryption). CSFs that are stored in the Resource Repository may be activated in a CSEE.

A CSEE may be pre-installed or downloaded by the host. A downloaded CSEE or pre-installed CSEE is required to be activated for use. A CSEE may support the ability to have additional CSEEs activated within it. A CSEE may have a CSF embedded within the CSEE. That CSF may be implicitly activated when the CSEE is activated on a CSE.

To be used, a CSF is required to be activated on a CSEE that has been activated on a CSE. This is called an activated CSF. The CSF performs only the defined operations (e.g., a specific eBPF program or compression) that are reported by the CSx (i.e.., the underlying operation is not changeable).

A CSF that calls an ordered sequence of CSFs is an aggregator CSF (see 4.6). An aggregator CSF may be downloaded or pre-installed in a CSx. An aggregator CSF calls activated CSFs that are on the same CSx as the aggregator CSF. The Activated CSFs may be activated on different CSEEs (e.g., an aggregator CSF may be running in an Operating System CSEE while the CSF that it executes may be in a hardware CSEE). Function Data Memory (FDM) is device

memory that is available for CSFs to use for data that is used or generated as part of the operation of a CSF. Allocated Function Data Memory (AFDM) is a portion of FDM that is allocated for one or more specific instances of a CSF operation. Any specific instance of a CSF operation shall only be allowed to access the AFDM allocated for that instance of a CSF operation. An aggregator CSF and all constituent CSFs shall only be allowed to access the AFDM allocated for that instance of the aggregator CSF operation. AFDM may be explicitly deallocated or may be deallocated on a power cycle or reset condition. As part of deallocating AFDM, the physical memory that was allocated to that AFDM should be cleared in order to prevent a subsequent instance of a CSF operation from accessing user data that is in that physical memory. On a reset, power cycle, or sanitize operation that deallocates AFDM, all FDM should be cleared to prevent any instance of a CSF operation from accessing user data that was not specifically stored in AFDM for that instance of a CSF operation.

The Resource Repository is a region of memory and/or storage located within the CSx that may contain images of CSFs and CSEEs that are available for activation. These CSFs and CSEEs are required to be activated in the CSE in order to be utilized.

A Computational Storage Processor (CSP) is a component that is able to execute one or more CSFs for an associated storage system without providing persistent data storage. The CSP contains CSRs and Device Memory. The mechanism by which the CSP is associated with the storage system is implementation specific.

A Computational Storage Drive (CSD) is a component that is able to execute one or more CSFs and provides persistent data storage. The CSD contains a Storage Controller, CSRs, Device Memory, and persistent data storage.

A CSD may function as a standard Storage Drive, with existing host interfaces and drive functions. As such, the system is able to have a storage controller with associated storage memory, along with storage addressable by the host through standard management and I/O interfaces.

A Computational Storage Array (CSA) is a storage array that is able to execute one or more CSFs. As a storage array, a CSA contains control software, which provides virtualization to storage services, storage devices, and CSRs for the purpose of aggregating, hiding complexity, or adding new capabilities to lower-level storage resources. The CSRs in the CSA may be centrally located or distributed across CSDs/CSPs within the array.

4.2 Discovery

4.2.1 CSx Discovery Overview

Discovery of CSxes is fabric dependent and is outside of the scope of this architecture.

4.2.2 CSR Discovery Overview

Once a CSx is discovered, to utilize Computational Storage Resources (CSRs), the characteristics of that CSx need to be discovered. This involves a CSR discovery process for each discovered CSx. The CSR discovery process discovers all resources available including CSEs, CSEEs, CSFs, and FDM.

Discovery of a CSE includes information of any activated CSEEs and any activated CSFs in those CSEEs.

CSEEs in the Resource Repository may be discovered and information about any CSFs pre-activated in those CSEEs is returned. CSEEs in the Resource Repository are required to be activated in order to be used.

CSFs in the Resource Repository may be discovered. CSFs in the Resource Repository are required to be activated in order to be used.

Section 0 shows an example discovery flow. The specifics of a CSR discovery process are defined in API specifications (e.g., SNIA Computational Storage API).

4.2.3 CSF Discovery and Configuration Example

Figure 4.2 shows an example flowchart of discovery and configuration of a CSF. This example assumes that each of the actions can be completed and that there are no errors. This is only one example of how configuration is able to be completed.

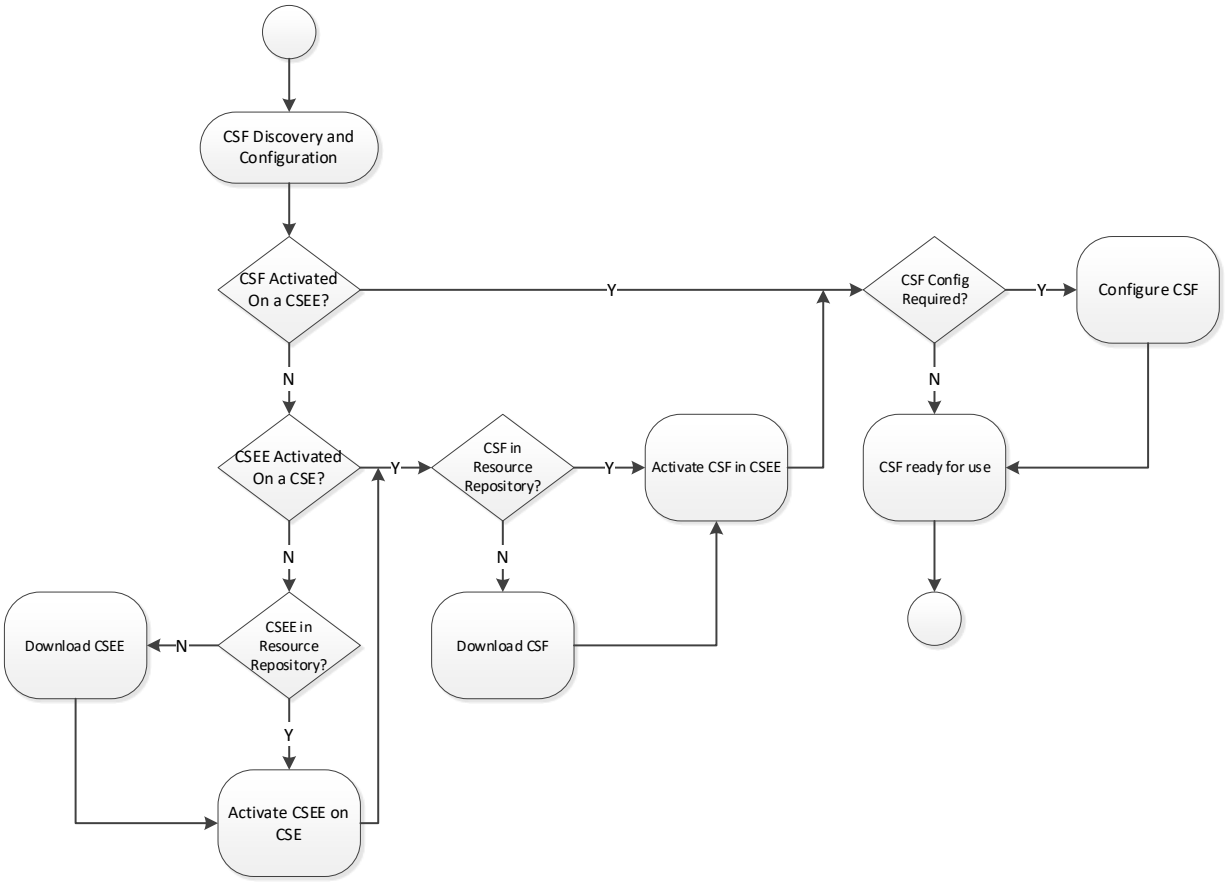

Figure 4.2 – CSF Discovery and Configuration Flowchart

For a CSF to be used, it has to be activated in a CSEE, so discovery and configuration of a CSF also includes discovery and configuration of a CSEE as well. The flow of the discovery and configuration process includes a number of steps to determine what is already activated in the CSx. For a CSF or CSEE that is not already activated, there is a discovery to determine if the desired CSF or CSEE exists in the Resource Repository. If a desired CSF or CSEE is not in the Resource Repository, then it has to be downloaded to the Resource Repository.

For a desired CSEE that is not activated, that CSEE is required to be activated in a CSE. After the desired CSEE is activated and the desired CSF is available in the Resource Repository, that CSF is activated in the CSEE.

For a desired CSF that is activated in a CSEE, that CSF is configured, if there are static configurations that are required for all executions of that CSF. Once the CSF is configured it is available for an application to execute.

4.3 Configuration

4.3.1 CSE Configuration Overview

Use of a CSE may require configuration to prepare it for use. One aspect of CSE configuration is activation of one or more CSEEs. A CSEE may be activated in the CSE at time of manufacture and therefore not be required to be activated as part of configuration. A CSEE that is activated in a CSE at time of manufacture may or may not be able to be deactivated. One implementation of a CSE configuration process is defined in the SNIA Computational Storage API.

4.3.2 CSEE Configuration Overview

A CSEE may be configured to prepare it for use. A CSEE is required to be activated in order to be used by a CSE. One aspect of CSEE configuration is activation of one or more CSFs. A CSF may be pre-activated in the CSEE and therefore not be required to be activated as part of configuration. A CSF that is pre-activated in a CSEE may or may not be able to be deactivated. The specifics of a CSEE configuration process are defined in API specifications (e.g., SNIA Computational Storage API).

4.3.3 CSF Configuration Overview

A Computational Storage Function may be configured to prepare it for use. A CSF is required to be activated in order to be used by a CSEE. The specifics of a CSF configuration process are defined in API specifications (e.g., SNIA Computational Storage API).

This process may be done once for the CSF, prior to any specific invocation of the CSF, or as parameters associated with the invocation of a CSF.

4.4 Security

4.4.1 General

Security requirements for computational storage vary significantly (e.g., depending on the environment, interconnectivity, and sensitivity of data). As such, security is presented in this document as considerations that may be used to help determine the security that is appropriate to the risks. Some of the considerations are written such that specific requirements are identified for certain elements of security (e.g., a decision to use encryption results in specific requirements such as security strength and key management).

The security considerations for a single tenant or host have been written with the following assumptions:

a) the environment consists of a single physical host or virtual host with one or more CSxes;
b) the host is responsible for the security of the ecosystem that the CSxes operate within; and
c) CSx security requirements are comparable to the security requirements common to SSDs/HDDs.

It is important to note that multi-host environments and CSxes that participate in the protection of data, result in significantly more complex security considerations and requirements.

The security considerations for a multiple tenant or host environment have been written with the following assumptions:

a) the environment consists of multiple physical hosts or multiple virtual hosts with one or more CSxes;
b) responsibility for the security of the ecosystem that the CSxes operate within is shared amongst the hosts in the ecosystem; and
c) CSx security requirements are comparable to the security requirements common to SSDs/HDDs in a multi-tenant environment.

4.4.2 Privileged Access and Operations

As stated in the assumptions in 4.4.1, the host is responsible for much of the security in a basic computational storage configuration. Much of this security involves privileged operations that are performed/executed by individuals (systems administrators or other privileged users) and entities (e.g., system applications) that have elevated privileges that are beyond normal users and entities. While the security associated with these operations and accesses are out of scope for this document, this security is critical to protecting data, resources, configuration, and state.

4.4.3 CSx Security Considerations

The security considerations identified in this sub-section are those considered relevant, given the assumptions stated in 4.4.1. Security is anticipated to be an important element of most computational storage implementations.

A rogue or broken CSF could consume all of the resources that other CSFs may need to operate. Therefore, mitigation mechanisms (e.g. verification of CSFs and sandboxing of CSFs) should be considered.

Unless other steps are taken to prevent it, the associated storage for the CSx is consumable by any and all CSFs.

4.4.3.1 CSx Sanitization

As part of any Sanitize operation:

a) all instances of CSF operations should be terminated;
b) all activated CSEEs and CSFs should be deactivated; and
c) all memory allocations associated with FDM should be removed and associated FDM cleared.

4.4.3.2 CSx Data at-rest Encryption

Data at-rest encryption is a commonly required feature of storage devices. Therefore, a CSx should implement the same data at-rest encryption as would be implemented on any storage device in a similar application.

A CSx may include the capability to encrypt data prior to recording the resulting ciphertext on storage media and decrypt ciphertext that has been recorded on storage media. When data at-rest encryption is implemented, the following should be provided:

a) strong symmetric encryption that provides a minimum of 128-bits of security strength to protect data (e.g., selection of encryption algorithms and modes of operations suitable for storage to be protected);
b) cryptographic keys that are only used for one purpose (e.g., do not use key-encrypting keys (i.e., key wrapping keys) to encrypt data or use data encrypting keys to encrypt other keys);
c) key management functionality (see 4.4.3.3) necessary for the data at-rest encryption;
d) capability to rekey the data with a different data/media encryption key (DEK/MEK) (i.e., reading the data, decrypting it with the old key, encrypting the data with a new key, and writing the new ciphertext).

Additional elements of data at-rest encryption implementations may include:

a) controls on the amount of data protected under a single key as well as within the established cryptoperiods;
b) proof/verification of encryption (enabled versus disabled);
c) cryptographic modules used to protect sensitive or regulated data should be validated using recognized security criteria (e.g. ISO/IEC 19790, ISO/IEC 15408, or NIST FIPS 140-3); and
d) archiving/escrowing the keys and keying material on key management servers.

The use of data at-rest encryption within a CSx has the following implications:

a) import/export compliance issues may affect the sale, distribution, and use of the CSx in certain jurisdictions; which may require specific licensing; and
b) data reduction technologies (e.g., compression and deduplication) are generally ineffective when applied to ciphertext.

4.4.3.3 CSx Key Management

Key management functionality is typically included in conjunction with data at-rest encryption (see 4.4.3.2) as opposed to a standalone capability.

A CSx may include key management capabilities to support encryption and decryption of data as well as cryptographic erase-based storage sanitization (see 4.4.3.4). When key management is implemented, the following should be provided:

a) cryptographic services that provide a minimum of 128 bits of security strength;

b) key generation with sufficient entropy (e.g., at least 256 bits of entropy input for AES-256) that uses the entire key space;
c) secure distribution of the keys (e.g., authentication key or KEK);
d) secure storage of keys and key material (e.g., with a hardware security module); and
e) secure, secondary storage for key backup/recovery.

Providing Key Management within a CSx has the following implications:

a) import/export compliance issues may affect the sale, distribution, and use of the CSx in certain jurisdictions; which may require specific licensing.

4.4.3.4 CSx Storage Sanitization

The controlled elimination of data in the form of storage sanitization is a commonly required feature of storage devices, therefore, it should be applied to CSxes. Failure to include storage sanitization may expose data to unauthorized access.

A CSx may include storage sanitization capabilities for controlled elimination of data. When storage sanitization is implemented, the following should be provided:

a) storage sanitization (i.e., media-based or logical sanitization) using clear or purge methods;
b) cryptographic erase (i.e., purge sanitization method in IEEE 2883) that ensures all copies of the encryption keys used to encrypt the target data are sanitized (see 4.4.3.2 data at-rest encryption and 4.4.3.3 key management); and
c) validation of sanitization operation outcomes.

Additional elements of storage sanitization implementations may include:

a) producing records (i.e., evidence) of sanitization operations that are able to serve as proof of sanitization; and
b) sanitization performed in conjunction with autonomous data movement.

Providing storage sanitization within a CSx has the following implications:

a) sanitization is not disrupted by firmware update, etc.

4.4.3.5 CSx Roots of Trust (RoT)

Roots of Trust (RoT) in the form of highly reliable hardware and software components that perform specific, critical security functions that provide a firm foundation from which to build security and trust are often included to support data at-rest encryption (see 4.4.3.2), key management (see 4.5.3.2), and attestation functionality.

If the CSx includes RoT and Chains of Trust (CoT), the following NIST SP 800-193, 4.1.1 requirements should be implemented:

a) the security mechanisms are founded in Roots of Trust (RoT);
b) if Chains of Trust (CoT) are used, a RoT serves as the anchor for the CoT;

c) all RoTs and CoTs are either immutable or protected using mechanisms which ensure all RoTs and CoTs remain in a state of integrity; and
d) all elements of the Chains of Trust for Update (NIST SP 800-193, 4.1.2), Detection (NIST SP 800-193, 4.1.3) and Recovery (NIST SP 800-193, 4.1.4) in non-volatile storage are implemented in platform firmware.

4.4.3.6 CSx Software Security

Software within a CSx may be in the following forms:

a) computational Storage Engine Environment (CSEE); or
b) computational Storage Functions (CSFs).

This software may be pre-installed or downloaded by the host.

A CSx may include a wide range of software security mechanisms, but the following should be provided:

a) verification of the integrity of downloaded CSx software (e.g., use of checksums to detect errors); and
b) validation that the code is coming from a particular source and that the code has not been altered or compromised by a third party (e.g., use of code signing).

4.4.4 Multi-tenancy Security Considerations

The security considerations identified in this sub-section are those considered relevant, given the assumptions stated in 4.4.1. Multi-tenancy security typically requires explicit trust relationships. When explicit trust relationships are required to be established between a Computational Storage Trust Element and another entity, the participating parties or entities are required to provide identification, authentication, authorization, and access control. For each of these capabilities, they may be available at one or more levels of the Computational Storage architecture (e.g., identification could be at the CSx level, while access control may be at the CSF level).

4.4.4.1 Identification

When explicit trust relationships are required to be established (e.g., for multi-tenancy), the participating parties or entities are required to identify themselves and recognize the identities of other parties and entities.

If a Computational Storage Trust Element includes identification capabilities, the following should be provided:

a) identity information to be presented to other parties or entities that is unique for the environment;
b) the ability to receive identity information from other parties or entities; and
c) the ability to manage identification information used by the Computational Storage Trust Element and those parties or entities that interact with that Computational Storage Trust Element.

Identification alone is insufficient to establish a trust relationship.

4.4.4.2 Authentication

When explicit trust relationships are required to be established, the participating parties or entities are required to verify their identities through authentication.

If a Computational Storage Trust Element includes authentication capabilities, then that Computational Storage Trust Element shall have completed identification and the following should be provided:

a) authentication information (credential) to be presented to other parties or entities that is unique for each trust relationship;
b) support for mutual authentication;
c) cryptographic protection for all stored and transmitted credentials;
d) traceability for all attempted authentications (i.e., successful and failed) with the Computational Storage Trust Element;
e) management of authentication credentials used by the Computational Storage Trust Element as well as credentials that may be presented to that Computational Storage Trust Element by parties or entities; and
f) support for external or third-party authentication.

Authentication may include implicit authorizations and is dependent on Identification. Successful authentication constitutes a basic trust relationship.

4.4.4.3 Authorization

When explicit trust relationships are required to be established, the participating parties or entities are required to be authorized to perform a given action on a specific resource.

If a Computational Storage Trust Element includes authorization capabilities, then that Computational Storage Trust Element shall have completed authentication and the following should be provided:

a) explicit mapping of permissions/privileges to parties or entities;
b) support for a least-privilege model (i.e., a subject should be given only those privileges needed for it to complete its task) (e.g., avoid root, admin, or superuser types of authorization);
c) support for sufficient granularity (e.g., roles or entity versus party) to address the use and management of the CSx;
d) separation of security privileges and non-security privileges; and
e) traceability for all authorization associated with security-oriented operations.

Authorization is dependent upon successful authentication and authorization establishes constraints on what a party or entity is permitted to do.

4.4.4.4 Access Control

When explicit trust relationships are required to be established, the participating parties or entities are required to be granted access (or disallowed access) for a given action on a specific resource.

If a Computational Storage Trust Element includes access control capabilities, the following should be provided:

a) enforcement of access to CSx operations and resources based on what parties or entities are authorized;
b) support for access control models (e.g., discretionary access control, mandatory access control, role-based access control, attribute access control);
c) support for the use of policy engines (e.g., enforcement points for zero trust); and
d) support for the use of external/delegated decisions.

4.5 CSF Usage

4.5.1 CSF Usage Overview

Once configured, a host may use the CSF with:

a) a direct usage model; or
b) an indirect usage model.

In the direct usage model, the host sends a computation request that specifies a CSF to execute on data in any of AFDM, device storage, or system memory and store the output data in any of AFDM, device storage, or system memory.

In the indirect usage model, a CSF is executed on data associated with a host storage request based on:

a) parameters in the storage request (e.g., a field that specifies a CSF to be executed);
b) the data locality (e.g., an address range for which a CSF is to be executed); or
c) the data characteristics (e.g., the size of the specified data).

For the indirect usage model that operates on data based on locality or characteristics, the Storage Controller is configured to associate a CSF with data locality or data characteristics prior to sending a storage request.

4.5.1.1 Direct CSF Usage Model

Figure 4.3 shows an example of the direct CSF usage model.

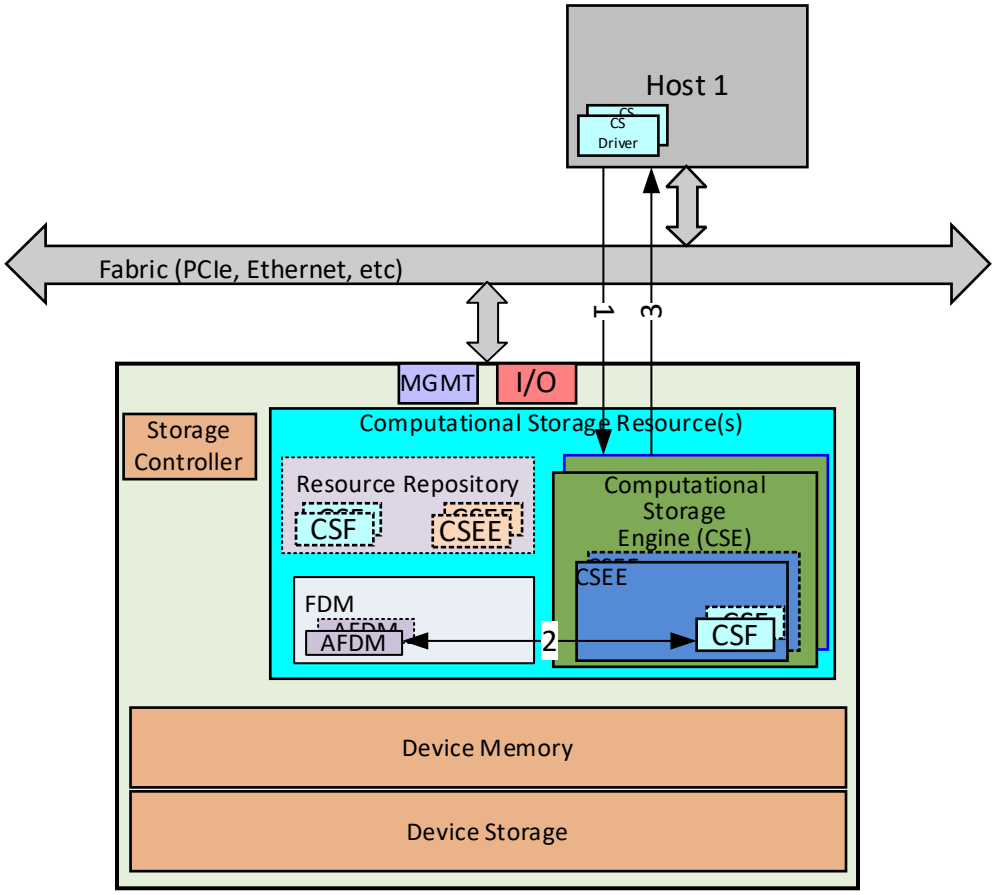

Figure 4.3 - Direct Usage Model

Figure 4.3 assumes that AFDM is allocated for the specific instance of the CSF and that data on which computation is to be performed is placed in that AFDM, prior to the request to the CSE. The result data, if any, is placed in the AFDM. The steps shown in Figure 4.3 for a direct usage model are:

(1) The host sends a command to invoke the CSF;
(2) The CSE performs the requested computation on data that is in AFDM and places the result, if any, into AFDM; and
(3) The CSE returns a response to the host.

4.5.1.2 Indirect CSF Usage Model

Figure 4.4 shows an example of the indirect CSF usage model.

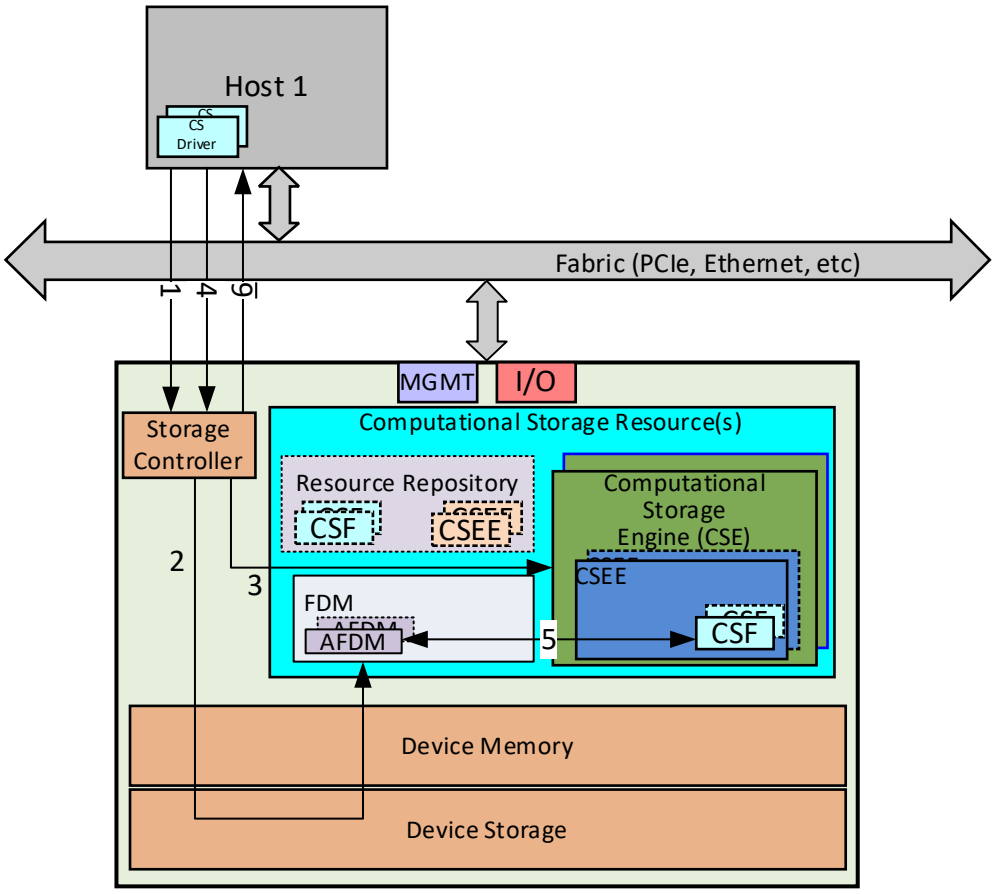

Figure 4.4 – Indirect Usage Interactions

Figure 4.4 assumes a read operation with computation on the data that is being read. The steps shown in Figure 4.4 to perform an indirect computation through the Storage Controller are:

(1) The host configures the CSD to associate a specific CSF with reads that have specific characteristics;
(2) The host sends a storage request to a Storage Controller where:
    a. that storage request is associated with that target CSF; and
    b. the storage controller determines what CSF is associated with the storage request;
(3) The Storage Controller moves data from storage into the FDM;
(4) The Storage Controller instructs the CSE to perform the indicated computation on the data in the FDM;
(5) The CSE performs the computation on the data and places the result, if any, into the FDM; and
(6) The Storage Controller returns the computation results, if any, from the FDM to the host.

4.5.2 CSF Execution

A CSF execution is specific to the type of CSF (e.g., for a compression CSF, a command may instruct the CSF to read from a given location in system memory, compress the data, and store the resulting data to a specified location in a storage device).

Execution of a CSF produces output and may require input and parameters. The input, optional parameters, and location of the output are required to be provided to the CSF as part of the execution request. Any of this information may be defined implicitly by the specific CSF. Examples of how this information is provided include:

a) Passing input location, output location, and parameters in the call to the CSF;
b) Placing input location, output location, and parameters in AFDM in a location defined by the CSF;
c) Passing parameters in the call to the CSF and placing input data in AFDM in a location defined by the CSF and returning output data as a return to the call to the CSF; or
d) Passing input location, output location, and parameters in a combination of the call to the CSF and in AFDM in a location defined by the CSF.

If information is placed in AFDM, a pointer to where that information is located is passed to the CSF or the information is placed in a location defined by the CSF (e.g., the first N words contain the input data location, the next M words contain the output data location, and the next P words contain the parameters). Input data or output data may each be located in device storage.

4.5.3 CSF Completion Output

Three types of information may be generated as the result of a CSF execution request:

a) CSF completion status;
b) Return value; or
c) Output data.

The CSF completion status is a mandatory output that indicates the successful completion or unsuccessful completion (i.e., completion that does not match expectations) of the requested CSF execution. CSF completion status is returned to the entity that requested the CSF execution.

The return value, if any, is a value returned from a computation (e.g., percentage of compression). The return value is returned to the entity that requested the CSF execution. The requirements for the return value are defined by the CSF.

Output data of a CSF, if any, is the result of the computation (e.g., compressed data). The output data may be placed in one or more destinations. Examples of destinations where the output data may be placed include:

a) AFDM;
b) Local Storage;

c) Host Memory; or
d) Other destinations that may be addressed by a host memory address (e.g., memory on another device).

4.5.4 CSF Unsuccessful Completion Handling

The CSEE or the host should handle unsuccessful completion (i.e., completion that does not match expectations) as indicated in the completion output from a CSF. The level of CSEE or host involvement in handling unsuccessful completions should be considered in a CSx architecture. Handling of an unsuccessful completion of a CSF by the CSEE may offload host handling of unsuccessful completions. However, a host may have increased visibility or resources to handle unsuccessful completions. Some CSF unsuccessful completions are only able to be handled by the host.

A CSEE may or may not have the ability to cleanup or rollback to a known state after an unsuccessful CSF completion. Host assistance with cleanup or rollback, if necessary, may be required. The CSEE or the host should check for security considerations (see section 4.4) that need to be addressed as a result of an unsuccessful CSF completion.

4.6 Sequencing

It may be desirable to execute multiple CSFs sequentially or in parallel to enable operations on data with minimal or no host involvement. The execution of this sequence of CSFs may be accomplished by the host initiating individual CSF executions or by an aggregator CSF that encapsulates or aggregates discrete CSFs. Any sequence of CSFs is possible within the constraints of the CSx resources. Any CSF, whether discrete or another aggregator CSF, may be called multiple times by an aggregator CSF.

4.6.1 Aggregator CSFs

The aggregator CSF invokes CSFs in a specified sequence. The constituent CSFs may be invoked sequentially, in parallel, or in combinations of parallel and sequential invocation. As each CSF in the specified sequence completes successfully, the aggregator CSF calls or invokes the subsequent CSF in that sequence. When all CSFs in the sequence complete, the aggregator CSF completes. This approach allows discrete CSFs to exist and be invoked independently or be combined into a sequence. Additionally, aggregator CSFs can call other aggregator CSFs to build complex sequences.

The location of input data, output data, and parameters for each CSF in an aggregator CSF may be defined by:

a) The application calling the aggregator CSF specifying where the input data, output data, and parameters of each constituent CSF are located;
b) The application providing resources for the aggregator CSF to manage where the aggregator CSF determines the location of input data, output data, and parameters for constituent CSFs within the provided resources;

c) The aggregator CSF managing resources provided at activation (i.e., explicit activation or implicit activation) where the aggregator CSF determines the location of input data, output data, and parameters for constituent CSFs within the provided resources; or
d) Any combination of the items in this list.

The aggregator CSF caller should obtain the appropriate security permissions for each CSF in the sequence before executing the aggregator CSF.

There are two types of aggregator CSFs:

a) Fixed sequence aggregator CSF; and
b) Variable sequence aggregator CSF.

A fixed sequence aggregator CSF executes a specific hard-coded sequence of CSFs. An aggregator CSF with different constituent CSFs or a different sequence of constituent CSFs is a different fixed sequence aggregator CSF. Figure 4.5 shows an example of a Fixed Sequence aggregator CSF that decrypts the input data, creates a list by searching and then sorting the data, and then encrypts that list to generate the output data.

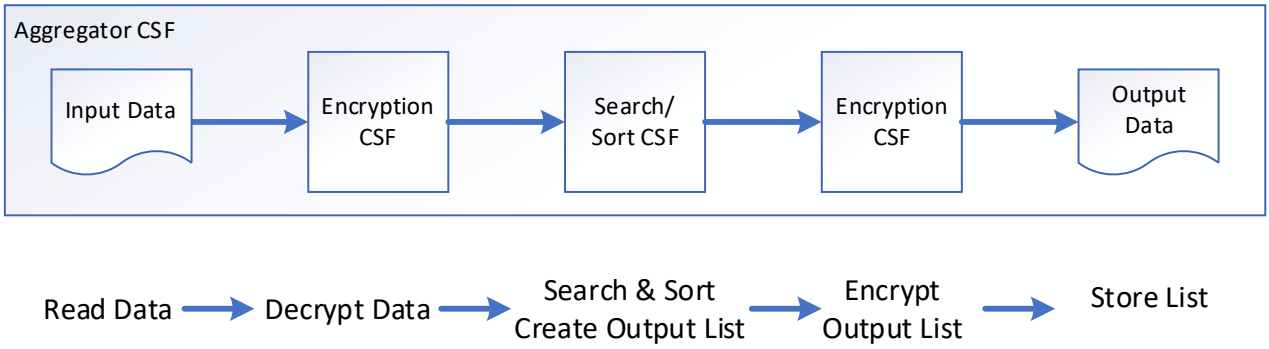

Figure 4.5 – Fixed Sequence Aggregator CSF

A fixed sequence aggregator CSF may include parallel execution of CSFs. Figure 4.6 shows an example of a fixed sequence aggregator CSF with parallel CSF execution. In this example, three CRC engines compute different CRC values in parallel on a single input data. Each CRC engine generates independent output data and each output may be placed in different destinations.

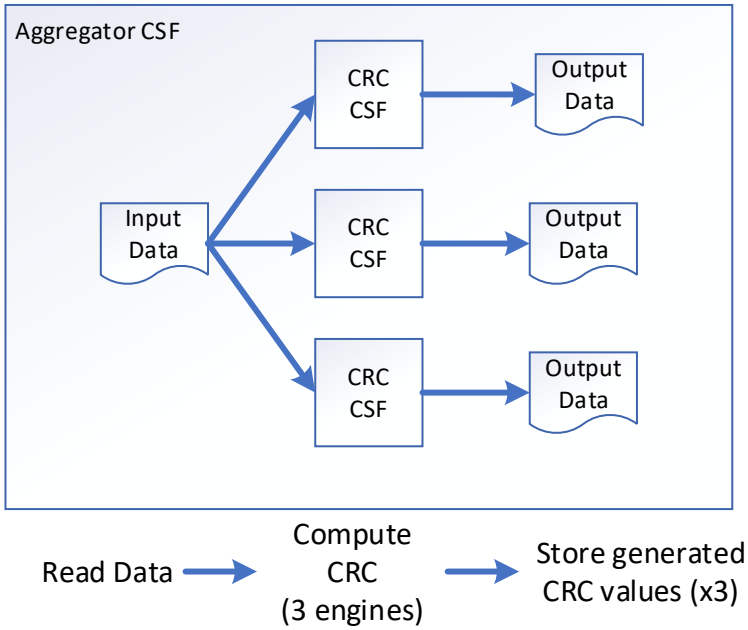

Figure 4.6 - Fixed Sequence Aggregator CSF with Parallel Execution

Fixed sequence aggregator CSFs may include parallel execution of CSFs that then feed into a subsequent execution of a CSF. Figure 4.7 shows an example of a fixed sequence aggregator CSF where a large file is broken into three chunks and each of the three chunks is fed into a search CSF. The results from the three chunks are processed by another search CSF to generate the output data. If additional decisions are made, parallel execution and variable sequences may be combined.

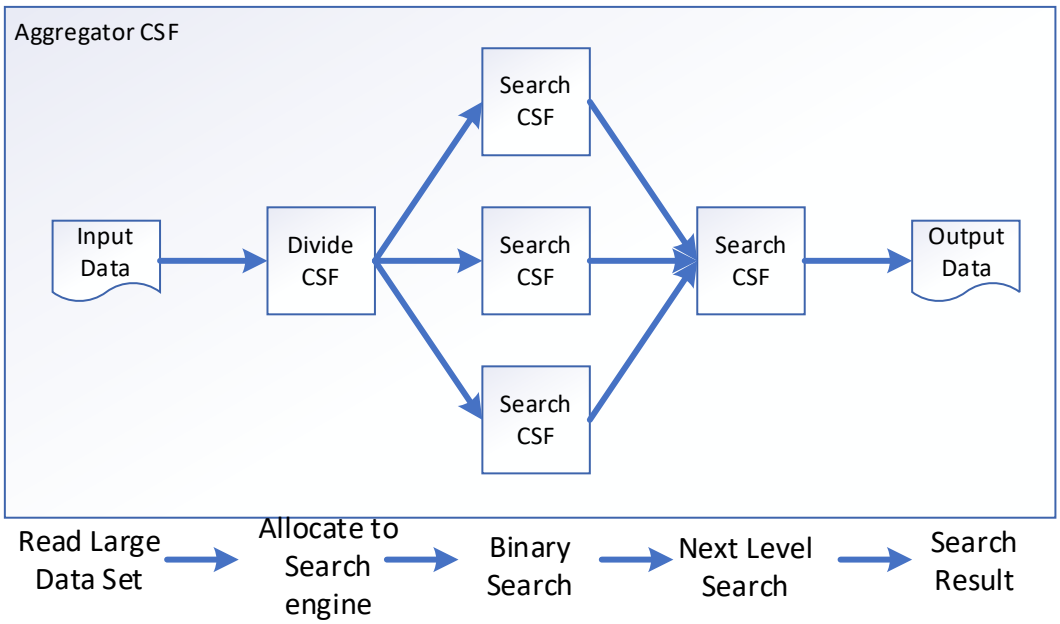

Figure 4.7 - Variable Sequence Aggregator CSF with Parallel Execution

A variable sequence aggregator CSF provides flexibility in the sequence of CSFs to execute. A variable sequence aggregator CSF may implement any sequence of CSFs, including parallel execution of CSFs and fixed sequence aggregator CSFs, specified by parameters that are provided at invocation. Figure 4.8 shows one example of a variable sequence aggregator CSF where the search/sort CSF is called repeatedly while the quantity of items found by the search CSF is greater than 20.

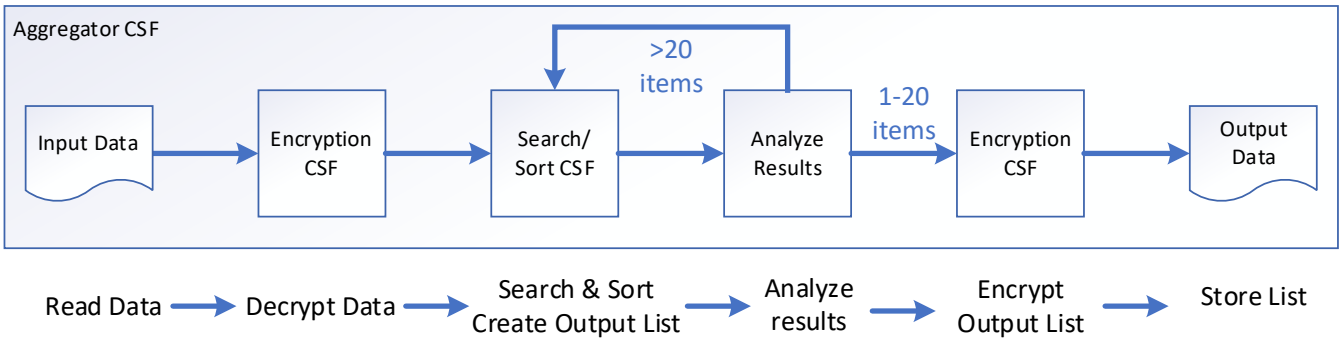

Figure 4.8 – Variable Sequence Aggregator CSF

Input location, output location, and parameters passed from one CSF to another CSF within the aggregator CSF may be passed from the host to the aggregator CSF or may be specified

by the aggregator CSF definition. Those locations and parameters that are passed from one CSF to another CSF in an aggregator CSF may be determined by the host and passed as parameters or may be provided by the aggregator CSF. Examples of how this information may be provided are described in section 4.5.2.

4.6.2 Aggregator CSF Completion Output

The aggregator CSF returns completion output as described in section 4.5.3. Completion output reflects the completion status, return value, and output of the entire sequence. In support of sending the sequence completion status, the aggregator CSF should monitor the completion status of each constituent CSF.

The completion output may include completion status for each constituent CSF. Completion output should provide sufficient detail to assist the host in handling unsuccessful completions, if any, (e.g., which constituent CSF(s) completed unsuccessfully in the sequence and supporting completion details).

4.6.3 Completion Handling

4.6.3.1 Host Involvement

The level of host involvement in processing completion outputs should be considered in a CSx architecture. A host may have increased visibility or resources to process completion outputs. For some completion outputs, the host may be the only entity capable of processing those outputs.

4.6.3.2 Aggregator CSF Involvement

An aggregator CSF may or may not be capable of processing completion outputs from the constituent CSFs.

The level of aggregator CSF involvement in processing completion outputs should be considered in the aggregator CSF architecture.

The aggregator CSF's processing of completion outputs may be controlled by parameters passed to the aggregator CSF. If an aggregator CSF is capable of and permitted to process completion outputs, results of completion outputs processed by the aggregator CSF may be reported to the host. An aggregator CSF may partially process completion outputs and partially rely on the host for processing completion outputs. When an aggregator CSF is unable to process a completion output from a constituent CSF, the aggregator CSF returns that information in a completion output to the host (e.g., the aggregator CSF may indicate an unsuccessful completion of the aggregator CSF, or the aggregator CSF may return the CSF completion output).

The description of an aggregator CSF should indicate its capabilities for processing completion outputs.

4.6.3.3 Successful Completion

An aggregator CSF may return a completion output indicating successful completion provided that all of the constituent CSFs have returned a completion output that matched expectations.

An example where an aggregator CSF completes successfully because constituent CSFs match expectations is:

1. the sequence includes a compression CSF followed by an encryption CSF;
2. a file is passed as input to the sequence;
3. the compression CSF executes (with the expectation that 50% or more compression will be achieved) and the completion output for the compression CSF reports that 60% reduction was achieved;
4. the aggregator CSF evaluates the compression CSF completion output and determines that the completion output does match expectations, therefore the sequence continues;
5. the encryption CSF executes on the output data that was generated by the compression CSF;
6. the aggregator CSF evaluates the encryption CSF completion output and determines that the completion output does match expectations, therefore the sequence completes successfully; and
7. the aggregator CSF may report the completion output from the constituent CSFs to the host.

An aggregator CSF may also return a completion output indicating successful completion if one or more constituent CSFs provides completion output that does not match expectations, but the aggregator CSF processes the completion output and determines that the completion output is acceptable. An example of this scenario is:

1. the sequence includes a decompression CSF followed by a search CSF;
2. a file is passed as input to the sequence;
3. the decompression CSF executes (with the expectation to decompress the file) and the completion output for the decompression CSF reports that the file is not compressed (no decompression performed);
4. the aggregator CSF evaluates the decompression CSF completion output and determines that the output does not match expectations, but it is acceptable to continue since the file is already decompressed;
5. the search CSF executes on the already decompressed file;
6. the aggregator CSF evaluates the search CSF completion output and determines that the output does match expectations, therefore the sequence completes successfully; and
7. the aggregator CSF may report the completion output from the constituent CSFs to the host.

4.6.3.4 Unsuccessful Completion

An aggregator CSF does not continue a sequence if a CSF returns a completion output that does not match expectations, and the aggregator CSF:

1. is not capable of continuing the sequence;
2. is not permitted to continue the sequence; or
3. determines it should not continue the sequence.

For example, a sequence includes two functions where the first CSF is decryption. The completion output reports that the file was not decrypted as expected (e.g., incorrect key). The completion output does not match expectations. The aggregator CSF evaluates the completion output and determines not to continue the sequence because the second CSF would be unable to execute on encrypted data.

Unsuccessful completion output from an aggregator CSF indicates unsuccessful execution of the aggregator CSF or on one of the constituent CSFs. If the aggregator CSF does not complete successfully, the completion status of each constituent CSF may require individual inspection to confirm or validate the completion status. Any aggregator CSF return value or output may also require validation.

An aggregator CSF that is unable to complete the sequence of CSFs, may or may not have the ability to cleanup or rollback to a known state. If cleanup or rollback is necessary, then the host or CSEE may need to assist with that cleanup or rollback. The host should look at any CSF in the sequence that does not complete successfully to determine if there are security issues that need to be addressed. For example, there may be data remnants as a result of the incomplete sequence that need to be cleaned up.

See section 4.5.3 for more information about CSF output.

5 Example Computational Storage Functions

This section describes example Computational Storage Functions (CSFs) (see 4.1).

See sections 4.2 and 4.3 for information about CSF discovery and configuration.

5.1 Compression CSF

A compression CSF reads data from a source location, compresses or decompresses the data, and writes the result to a destination location.

A CSF configuration specifies the compression algorithm and associated parameters.

A CSF command specifies the source address and length and the destination address and maximum lengths.

5.2 Database Filter CSF

A database filter CSF reads data from source location(s), performs a database projection (column selection) and filter (row selection) on the data according to projection and filter conditions, and writes the result(s) to destination location(s).

A CSF configuration specifies the database format, table schema, selection and filter conditions, and associated parameters.

A CSF command specifies the source address and length, and the destination addresses and lengths.

5.3 Encryption CSF

An encryption CSF reads data from a source location, encrypts or decrypts the data, and writes the result to a destination location.

A CSF configuration specifies the encryption algorithm, keying information, and associated parameters.

A CSF command specifies the source address and length, and the destination address and length.

5.4 Erasure Coding CSF

An erasure coding CSF reads data from source location(s), performs an EC encode or decode on the data, and writes the result(s) to destination location(s).

A CSF configuration specifies the EC algorithm and associated parameters.

A CSF command specifies the source address and length and the destination addresses and lengths.

5.5 RegEx CSF

A regex CSF reads data from source location(s), performs a regular expression pattern matching or transformation on the data, and writes the result(s) to the destination location.

A CSF configuration specifies the RegEx string(s) and associated parameters.

A CSF command specifies the source address and length, and the destination address and length.

5.6 Scatter-Gather CSF

A Scatter-Gather CSF reads data from set of source location(s) and writes the data to a set of destination location(s).

A CSF configuration does not have any parameters.

A CSF command specifies the source addresses and lengths, and the destination addresses and lengths.

5.7 Pipeline CSF

A Pipeline CSF performs a series of operations on data according to a data flow specification, allowing different CSF commands to be combined together in a standardized way.

A CSF configuration does not have any parameters.

A CSF command specifies a collection of commands, their order and dependencies, and calculations defining the relationships of the addresses between commands.

5.8 Video Compression CSF

A video compression CSF reads data from a source location, compresses or decompresses the video, and writes the result to a destination location. In order to accommodate multiple parallel compressions, the video compression CSF may support a single compression stream or multiple compression stream

A CSF configuration specifies the stream, compression algorithm and associated parameters.

A CSF command specifies the stream, source address and length, and the destination address and maximum lengths.

5.9 Hash/CRC CSF

A hash/CRC CSF reads data from a source location, calculates a hash or CRC value based on the source data, and writes the result to a destination location.

A CSF configuration specifies the hashing/CRC algorithm and associated parameters.

A CSF command specifies the source address and length and the destination address.

As an optional feature, this CSF may calculate the hash/CRC value based on the source data and compare the hash/CRC result to a pre-calculated value supplied by the initiator. The CSS will notify the initiator whether the calculated value matches the supplied value.

5.10 Data Deduplication CSF

A data deduplication CSF reads data from source location(s), performs deduplication or duplication on the data, and writes the result(s) to the destination location(s).

A CSF configuration specifies the data deduplication algorithm and associated parameters.

CSF command specifies the source address and length, and the destination address and maximum lengths.

5.11 Large Data Set CSFs

This example is for a large dataset wherein the data is sharded into chunks that have semantic meaning to the application and are stored across a set of CSxes. To act on that data, it is necessary to map the data shards to the CSx where they are stored and then download a CSF to each of the devices. That CSF is then able to be executed in the CSx against each of the dataset's shards stored in that CSx. This may be done simultaneously on thousands of CSxes.

6 Example Computational Storage Execution Environment

This section describes example Computational Storage Execution Environments (CSEEs) (see 4.1).

See 4.2 and 4.3 for information about CSEE discovery, configuration, and activation.

6.1 Operating System CSEE

An Operating System CSEE provides a specific operating system environment (e.g., Linux). The Operating System CSEE may contain one or more activated CSFs and may support the activation of one or more downloaded CSFs.

6.2 Container Platform CSEE

A Container Platform CSEE provides an environment to host one or more Container CSEEs.

In order to provide CSFs, it is necessary to have this type of CSEE configured with a Container CSEE.

6.3 Container CSEE

A Container CSEE provides a container environment. The Container CSEE may contain one or more activated CSFs and may support the activation of one or more downloaded CSFs.

6.4 eBPF CSEE

An extended Berkeley Packet Filter (eBPF) CSEE provides an environment for running eBPF programs. The eBPF CSEE may contain one or more activated eBPF CSFs and supports the activation of one or more downloaded eBPF CSFs.

6.5 FPGA Bitstream CSEE

A FPGA Bitstream CSEE provides an environment for an FPGA device. The FPGA Bitstream CSEE may contain one or more activated CSFs and may support the activation of one or more downloaded CSFs.