TRUSTED® COMPUTING GROUP

SPECIFICATION

DICE Protection Environment

Version	1.0
Revision	0.13
January 17, 2024	

Contact: admin@trustedcomputinggroup.org

Public Review

Work in Progress

This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.

DISCLAIMERS, NOTICES, AND LICENSE TERMS

THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

CHANGE HISTORY

REVISION	DATE	DESCRIPTION
1.00/0.1	December 15, 2021	Initial draft
1.00/0.2	June 23, 2022	Final draft – Technical content complete
1.00/0.3/0.4	September 22, 2022	Add session migration, changes in response to comments/feedback
1.00/0.5	January 9, 2023	Multipart cmd/rsp, cleanup based on WG discussions
1.00/0.6	January 26, 2023	Changes in response to TC feedback
1.00/0.7	May 25, 2023	Add localities and recursive destroy, removed session migration, various changes in response to comments/feedback
1.00/0.8	August 17, 2023	Revisions based on feedback
1.00/0.9	August 25, 2023	Multipart/locality revisions
1.00/0.10	November 1, 2023	Resolve TC feedback
1.00/0.11	November 9, 2023	last minute corrections
1.00/0.12	December 16, 2023	Address feedback before review

CONTENTS

1 SCOPE

This specification defines requirements for a DICE Protection Environment (DPE) and DPE Profiles.

1.1 Key Words

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this document normative statements are to be interpreted as described in RFC-2119, Key words for use in RFCs to Indicate Requirement Levels.

1.2 Statement Type

Please note a very important distinction between different sections of text throughout this document. There are two distinctive kinds of text: informative comment and normative statements. Because most of the text in this specification will be of the kind normative statements, the authors have informally defined it as the default and, as such, have specifically called out text of the kind informative comment. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as of the kind informative comment, it can be considered a kind of normative statement.

EXAMPLE:

Start of informative comment

This is the first paragraph of 1–n paragraphs containing text of the kind *informative comment* ...

This is the second paragraph of text of the kind *informative comment* ...

This is the nth paragraph of text of the kind *informative comment* ...

To understand the TCG specification the user must read the specification. (This use of MUST does not require any action).

End of informative comment

2 REFERENCES

[1] Trusted Computing Group, "Hardware Requirements for Device Identifier Composition Engine Level 00, Revision 78," 22 March 2018. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/Hardware-Requirements-for-Device-Identifier-Composition-Engine-r78_For-Publication.pdf.

[2] Trusted Computing Group, "DICE Layering Architecture Version 1.0 Revision 0.19," 23 July 2020. [Online]. Available: https://trustedcomputinggroup.org/resource/dice-layering-architecture/.

[3] Trusted Computing Group, "DICE Attestation Architecture Version 1.00 revision 0.23," 1 March 2021. [Online]. Available: https://trustedcomputinggroup.org/resource/dice-attestation-architecture/.

[4] Trusted Computing Group, "DICE Certificate Profiles Version 1.0 Revision 0.01," 23 July 2020. [Online]. Available: https://trustedcomputinggroup.org/resource/dice-certificate-profiles/.

[5] Trusted Computing Group, "TCG Glossary Version 1.1 Revision 1.0," 11 May 2017. [Online]. Available: https://trustedcomputinggroup.org/resource/tcg-glossary/.

[6] National Institute of Standards and Technology, "SP 800-133 Recommendation for Cryptographic Key Generation," 2020. [Online]. Available: https://csrc.nist.gov.

[7] National Institute of Standards and Technology, "SP 800-57 Part 1 Rev. 5 Recommendation for Key Management: Part 1 – General," May 2020. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final.

[8] T. Perrin, "The Noise Protocol Framework," July 2018. [Online]. Available: https://noiseprotocol.org/noise.html.

[9] Internet Engineering Task Force, "Concise Binary Object Representation (CBOR)," December 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8949.

[10] Internet Engineering Task Force, "Concise Data Definition Language (CDDL) A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures," June 2019. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8610.

[11] Internet Engineering Task Force, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," May 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5280.

[12] Internet Engineering Task Force, "CBOR Object Signing and Encryption (COSE)," July 2017. [Online]. Available: https://datatracker.ietf.org/doc/rfc8152/.

[13] Internet Assigned Numbers Authority, "Named Information Hash Algorithm Registry," September 2016. [Online]. Available: https://www.iana.org/assignments/named-information/named-information.txt.

[14] National Institute of Standards and Technology, "SP 800-160 Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems," 2020. [Online]. Available: https://csrc.nist.gov.

[15] Internet Engineering Task Force, "Hybrid Public Key Encryption," February 2022. [Online]. Available: https://datatracker.ietf.org/doc/rfc9180/.

3 TERMS AND DEFINITIONS

For the purposes of this specification, the following terms and definitions apply. This specification assumes the reader is familiar with the TCG DICE specifications [1] [2] [3] [4] and common Trusted Computing terminology, as defined in [5].

3.1 Acronyms

ABBREVIATIONS	DESCRIPTION
CDI	Compound Device Identifier
DICE	Device Identifier Composition Engine [1]
DPE	DICE Protection Environment
ECA	Embedded Certificate Authority
HKDF	HMAC-based Key Derivation Function
IP	Intellectual Property, e.g., IP Block
IPC	InterProcess Communication
MCU	Microcontroller Unit
RPC	Remote Procedure Call
SPI	Serial Peripheral Interface
SVN	Security (-relevant) Version Number
TCB	Trusted Computing Base, see also [5]
TCG	Trusted Computing Group

3.2 Nomenclature

This specification uses the terms *child* and *parent* to describe the relationship between components in a DICE layered architecture. A parent component invokes or spawns a child component. In some cases, a parent component can have multiple child components, and a child may itself be a parent and spawn additional child components. In graph theory terms, a DICE layered architecture can be viewed as an arborescence.

4 INTRODUCTION

Start of informative comment

This document specifies the command interface, behavior, and profile requirements of a DICE Protection Environment (DPE). A DICE Protection Environment protects DICE-related secrets and helps enforce DICE-related policies. In a layered DICE architecture (see [2]) a component employs DICE without a DPE by directly handling and processing Compound Device Identifier (CDI) values. CDI values are very sensitive and are as long-lived as the components they represent. With a DPE, instead of handling DICE secrets directly, a component employs DICE by issuing commands to the DPE. The DPE retains possession of secret values (e.g., CDIs and private keys) and does not expose sensitive data to any component or *client*. Instead, the DPE provides each client with a *context handle* that a client uses to refer to the data corresponding to the DICE operations initiated by that client.

Context handles are opaque to clients. The value of a context handle may be a simple index to a structure containing CDI values and other data stored by the DPE, or it may be the actual CDI values and other data that are encrypted so it can only be decrypted by the DPE. A context handle is less sensitive than a CDI itself because it is ephemeral and useful only under limited conditions. A context handle corresponds to only one set of CDI values for a specific component and, when a context handle is destroyed, the associated CDI values are also destroyed. Communication between a client and a DPE occurs in a *session* that might be encrypted. Context handles are one-time-use and bound to a single session. Commands that consume a context handle invalidate that handle and generate a new handle for the subsequent command, if appropriate.

Use of a DPE may reduce the following risks:

- CDI exfiltration by exploiting hardware, firmware, or software vulnerabilities or side channels
- CDI leakage due to normal system memory management such as swap or hibernation
- implementation-specific issues, including cryptographic algorithm implementations, across heterogeneous components

A DICE-based system may also be able to improve performance by offloading DICE computations to a DPE. A DPE implementation may perform computations asynchronously or otherwise employ caching techniques, provided the DPE complies with this specification. For example, a DICE-based system may have booted multiple components before DICE computations have completed for the first component within the DPE.

DICE layering semantics and CDI derivations are the same, or can be the same, whether a DPE is used or not. Using a DPE does allow for additional DICE features but a direct translation of any existing DICE-based system to an equivalent system that uses a DPE should be possible. In addition to protecting DICE CDIs, a DPE also protects keys derived from CDIs like private signing keys and sealing keys, as these are part of a client's DPE context. However, using a DPE does not fully compensate for potential vulnerabilities in device firmware.

A DPE is not stateless. It tracks all valid context handles, sessions, and the mappings between them. The number of contexts supported by a DPE is implementation specific. In its simplest form, a DPE supports one context and one plaintext session.

This specification does not define or limit how a DPE can be implemented. Examples of environments that could be used for a DPE implementation are a secure coprocessor, discrete secure hardware, a Trusted Execution Environment (TEE), a type-1 hypervisor, operating system kernel, or another type of environment isolated with a hardware-backed mode switch. A DICE-based system could also transition to using a DPE at a particular layer.

Because of this flexibility of implementation, no specific hardware requirements are defined in this specification. Some level of hardware-backed protection is required to isolate the DPE environment and make it useful, but the nature or strength of that isolation is not defined or constrained here. For example, the hardware-backed protection may be a Memory Management Unit (MMU) for an OS kernel, or a separate IP block or Microcontroller Unit (MCU).

Notably a DPE can be implemented without any persistent storage, true random number generator (TRNG), or a real time clock. However, a DPE implementation may well have such capabilities.

End of informative comment

5 CONCEPTS

This section provides details on the core concepts of a DICE Protection Environment.

5.1 DPE and DICE Layering

Figure 1 illustrates the difference between a simple layered DICE flow with and without help from a DPE. The diagram is simplified to illustrate the main goal of the DPE: to protect CDI values. The construction of Layer 0 from a UDS, destruction of CDI values, certificates, and other aspects of layered DICE architecture are not shown. Destroyed CDIs are illustrated with a surrounding dotted line.

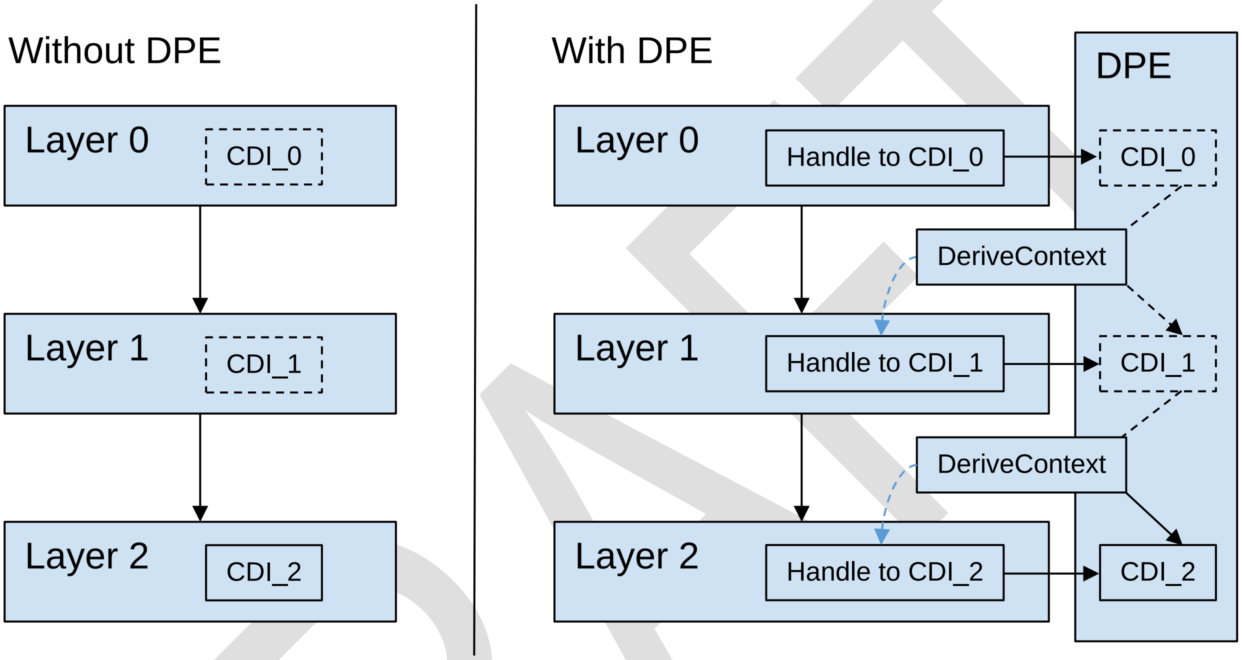

Figure 1: DICE layering with and without a DPE

5.1.1 Example Flow

The following simplified pseudocode flow illustrates a system with two firmware components using a DPE to perform basic attestation using a signature. Session handshake and encryption details are omitted.

```
// ROM code
session = dpe.OpenSession()
context = dpe.InitializeContext(uds)
context = dpe.DeriveContext(context, firmware1_hash)
Run(firmware1, session, context)

// Firmware1 code
context = dpe.DeriveContext(context, firmware2_hash, allow-new-context-to-derive=false)
Run(firmware2, session, context)

// Firmware2 code
```

```
context, cert_chain = dpe.GetCertificateChain(context)
context, leaf_cert = dpe.CertifyKey(context)
signature = dpe.Sign(context, attestation_challenge)
attestation_response = leaf_cert + cert_chain, signature
```

5.2 Use Cases

A DPE MUST support at least one of the following use cases (see [3] [5]):

- Signing – To wield signing keys derived and/or certified using a layered DICE architecture.
- Sealing – To wield decryption keys derived using a layered DICE architecture.

5.3 Clients

Clients of a DPE can be in any system that uses a layered DICE architecture for signing or sealing (see [2]). A client owns a DPE session endpoint and owns DICE-related information that it delegates to the DPE for safekeeping. The scope of a client is bounded by access to the DPE session endpoint and context handle. A client wields a session endpoint and context handle to interact with the DPE. In most cases, a client corresponds directly to a DICE Trusted Computing Base (TCB) component: see the DICE Layering Architecture specification [2]. However, this specification does not define or constrain the nature of a client, how a client manages session endpoints and context handles, or the communication path between it and a DPE. Calling a DPE may involve an IPC/RPC mechanism, a SPI interface, or even just a simple function call.

If a DPE uses encrypted sessions, the client that opens the session is responsible for authenticating the DPE. This may require the initial client to store static identity data (e.g., a public key) for the DPE in such a way that it is available for authenticating the new session.

Start of informative comment

An initial client (usually the platform RTM) may be responsible for authenticating a DPE before subsequent components are allowed to execute. For example, this authentication may be performed using a pre-provisioned public key corresponding to the DPE identity. Authentication of the DPE only needs to be done once per session (see `OpenSession` in section 6.2).

End of informative comment

5.4 Configuration and Profiles

This specification is flexible in order to accommodate a variety of possible implementations. In contrast, a DPE implementation does not need to be flexible and is expected to have a fixed configuration and fixed capabilities. It is expected that capabilities and configuration of a DPE will be determined during system design and/or integration phases, not at runtime. However, a DPE implementation may offer configurability outside the scope of this specification, for example, with vendor-specific commands.

To promote compatibility and interoperability, DPE implementations MUST conform to a profile. System integrators need to ensure that a DPE and its clients use the same profile. A DPE profile is *complete*, that is, it specifies all attributes that this specification leaves up to implementors. This specification lists all attributes that a DPE profile is required to specify: see section 7.2.

5.5 Version Compatibility

The versions of this specification are mutually compatible across minor versions from the view of a client. Changes that introduce incompatibility from the view of the client will be accompanied by a new major version. Minor version changes introduce requirements on DPE implementations only if the contract with the client is unaffected. In other words, there should be no reason for a client to differentiate between different minor versions if the major version is the same.

5.6 Contexts

A DPE context comprises all the DICE-related information for a particular component, i.e., client. A DPE context contains, at least, a CDI value, but in practice there is usually additional information. A DPE uses a context handle to refer to (or to contain) DPE context for a given client. Context handles are opaque to DPE clients. Context handle values may be a handle, an index into context data held internally by the DPE, or the context handle value may itself contain the client's context data encrypted in a way that only the DPE can decrypt it. While a DPE implementation has flexibility in how it constructs context data and handles, it has the following constraints:

1) The context handle MUST be unguessable in practice. If the context handle value is an index to a client's DPE context data, it SHOULD be random and at least 16 bytes in length. The reason for this is that a context handle authorizes operations on the associated context. So, for example, it's possible for parent and child components to share the same encrypted session, but the child should not be able to leverage that shared session to impersonate the parent.
2) If the context handle value is encrypted data, it MUST use an algorithm with at least 128 bits of security strength and it MUST be integrity protected. Security strength is used here as defined in NIST SP800-133 [6] and NIST SP800-57 [7].
3) The context handle MUST comply with size limits imposed by the profile, if any.
4) The context handle MUST be bound to a specific session and a specific locality.
5) The context handle MUST NOT remain valid after it has been used by a command. In other words, context handles are single use. New context handles are returned by a DPE within a response to a client so it can be used on a subsequent command. Once a context handle is provided to the DPE by a client, the context handle is invalidated by the DPE.

Multiple contexts can be bound to the same session by invoking `InitializeContext` multiple times or by invoking the `DeriveContext` command with `retain-parent-context` set to true and the `new-session-initiator-handshake` argument omitted.

5.6.1 Default Contexts

DPE client sessions may not require or benefit from multiple distinct DPE contexts. In these scenarios, the default context may be used. The only distinguishing characteristic of a default context is that there is no client-visible context handle associated with the default context. A default context is stored internally to the DPE and is indicated in a command by omitting the context handle argument. The primary benefit of using the default context is that a client is not required to keep track of or provide context handles.

A DPE SHOULD support default context(s) and may support only default context(s). If a DPE supports default contexts, it MUST support one default context per session. A DPE MUST NOT allow simultaneous use of a default context and context handles within the same session: these are mutually exclusive.

A default context can be destroyed like any other context. If a client wishes to transition from a single context session to a multiple context session, the client can use the `RotateContextHandle` command to transition the default context to a context handle. A DPE MUST return an `invalid-argument` error in response to any command that attempts to use a default context after it has been destroyed (by calling `DestroyContext`) or rotated (by calling `RotateContextHandle`) until the default context is explicitly initialized again by the client.

A default context can be initialized by setting the `use-default-context` argument to true for the `InitializeContext` command. A DPE can support an automatic initialization procedure for default contexts. No initialize command is required for an automatic initialization procedure and clients can proceed directly to other commands that require an initialized context.

5.6.2 Simulation Contexts

A DPE may support simulation contexts. A simulation context is used the same way as a normal context but the DPE disallows operations that use private keys: this includes commands like `CertifyKey` or `Unseal`. Using a simulation context allows a client to derive public keys and certificates associated with different inputs than were used to boot the currently running system.

Start of informative comment

The primary use case for a simulation context is referred to as *predictive sealing*. For example, if a client component anticipates an update or expects to make a change that would result in a new CDI value for a subsequent component, that client can use a simulation context to seal data to the expected future measurement (i.e., the predicted measurement) of the subsequent component. This works because a DPE will allow sealing and public key derivation but will not allow unsealing for a simulation context.

End of informative comment

5.6.3 Context Initialization

A DPE profile specifies how the initial state of a context is derived. This derivation process is the same whether it is executed as part of an `InitializeContext` command or automatically when a DPE starts. At the end of the initialization process, the new context contains a UDS or CDI(s). The `InitializeContext` command has a `seed` argument that allows the client to provide an input to the derivation.

The following list comprises examples of how a DPE profile might be initialized:

- The DPE might use the seed argument directly as a UDS or CDI value.
- The DPE might use a UDS it has access to internally, ignoring or mixing in the seed argument.
- The DPE might use its own internal CDI and certificate chain as the initial state, ignoring or mixing in the seed argument.
- The DPE might expect the seed to be a structured type providing multiple CDI values and certificate data.

If an initialization process involves a DICE UDS, the DICE hardware requirements [1] SHALL apply.

A DPE that supports simulation contexts might retain the `InitializeContext` seed argument value in association with the current session to enable the subsequent initialization of simulation contexts. A DPE MUST NOT use the seed for any other purpose.

A DPE MUST prevent further access via DPE commands to UDS, CDI, or other seed value(s) used in initialization until the next system reset. A DPE MAY retain the value(s) in association with the current session for initializing simulation contexts. In these cases, a DPE MUST NOT use the retained value(s) for any other purpose. If a subsequent `InitializeContext` operation attempts to use the internal value again to initialize a context that is not a simulation context, a DPE MUST abort the operation and return an `initialization-seed-locked` error.

5.7 Sessions

A DPE client invokes commands within a session. A DPE keeps a record of all open sessions and each context handle produced by a DPE is bound to a single session. The DPE maintains the bookkeeping for and enforces this binding. For example, if a context handle received by a client in one session is used in another session, the DPE MUST reject the context handle as invalid. In other words, sessions are independent from each other.

The command/response messages within a particular session are ordered and serialized, e.g., *command1*, *response1*, then *command2*, *response2*, and so on. Across different sessions, commands may be interleaved or concurrent. If a command or response message is not delivered, the session can be brought back in sync with the `SyncSession` command.

5.7.1 Encrypted Sessions

A session may be encrypted or plaintext. Encrypted sessions are designed for systems where communication between a client and DPE passes outside of the client's Trusted Computing Base (TCB). A DPE MUST support a plaintext session with a session ID of zero and can support any number of encrypted sessions. A DPE MUST NOT support more than one plaintext session per locality (see section 5.9 Localities). If encrypted sessions are supported for a locality, a DPE MUST use an encrypted session for all commands on that locality except `OpenSession`, `GetProfile`, and `SyncSession`. Encrypted sessions are created using the `OpenSession` command, or as part of a `DeriveContext` command. A session can be closed using the `CloseSession` command and doing so destroys all context data associated with the session.

If encrypted sessions are supported, the Noise_NK_25519_AESGCM_SHA256 protocol [8] SHOULD be supported. Support for other protocols is allowed, but this specification anticipates this type of protocol. The number of encrypted sessions supported and the session protocol, including the format of the handshake and encrypted messages, is specified by a DPE profile. When using a protocol that specifies a maximum number of messages per session, say due to key exhaustion concerns or counter maximums, etc., a DPE MUST fail all commands on an exhausted session with a `session-exhausted` error code (see section 5.10.4).

Protocols used for DPE encrypted sessions MUST have the following security properties:

- Confidentiality: payload data is encrypted
- Integrity: payload data includes protection against tampering
- Authenticated: clients can authenticate the DPE against a known public key or pre-shared secret key. A DPE is not expected to authenticate clients.

Protocols used by a DPE for encrypted sessions SHOULD have the following security properties:

- Key independence: forward and backward secrecy
- Privacy: neither client nor DPE can be identified by observing handshake and/or transport messages

A DPE that supports encrypted sessions has an identity associated with a public key that can be authenticated by clients. The provisioning, rotation, and storage of the identity is implementation-dependent and out of scope for this specification. For example, this may be a static identity that is provisioned in the factory and remains the same for a DPE's lifetime, or it may be randomly generated when a software DPE is instantiated and provisioned to clients via an Operating System service. The DPE identity SHOULD be unique per DPE instance.

When using Noise_NK_25519_AESGCM_SHA256 [8] the following requirements apply:

1) A DPE SHALL be the responder for every negotiation and SHOULD NOT authenticate the initiator. In other words, the DPE accepts commands from any client, regardless of the nature and identity of the client.
2) The client that invokes `OpenSession` is responsible for authenticating the DPE's identity (see section 6.2).

Start of informative comment

Typically, the DPE identity (i.e., public key) is already known to the client by some other means.

End of informative comment

3) A DPE SHALL maintain ordering of messages in a session and MUST support the `SyncSession` command to get back in sync if necessary.
4) New sessions created as part of a `DeriveContext` command use the current session's binding token as a pre-shared key in the new session negotiation. Noise_NNpsk0_25519_AESGCM_SHA256 SHOULD be supported.

Other protocols used by a DPE to implement encrypted sessions should also meet the requirements for Noise_NK_25519_AESGCM_SHA256 where applicable.

5.8 Interface

There are two categories of interface for a DPE: (1) message-based, or (2) direct. A DPE implementation MUST support at least one interface.

A message-based interface involves command and response message pairs, where each message is encoded as described by this specification. Transport of the message is implementation-specific and not constrained. For example, messages may traverse a network, a bus, or a pipe.

A direct interface involves implementation-specific direct invocation of commands: the nature of the interface is not constrained. For example, a direct interface could be an API in any programming language.

Start of informative comment

Note that the required commands and functionality implemented by a DPE are not influenced by the interface definition. The interface definition is simply the mechanism by which a client interacts with a DPE.

Further, it is strongly recommended that, for any direct DPE interface, there is a clear mapping between the implementer-defined interface and the message-based interface defined in this specification. This mapping is typically referred to as a *translation layer.*

The goal of providing a translation layer for any direct DPE interface is to promote interoperability and to allow a message based DPE test harness to work with and validate any DPE implementation.

End of informative comment

Since a direct interface for a DPE is implementation-specific, the remainder of this specification discusses only the message-based interface.

5.9 Localities

DPE clients are modeled as existing within a *locality*. A DPE implementation can support a single locality or many. This specification does not constrain or define what a locality is or does beyond the scope of a few specific requirements. For example, an implementation may use localities to distinguish between co-processors, execution levels within a processor, or something else.

If a DPE supports multiple localities it is expected to be able to distinguish between them based on information not available as part of the DPE command interface. Clients do not indicate their locality to a DPE as part of a command. The locality of a client when a command is invoked is referred to as the current locality.

Context handles are bound to a locality and can only be used by a client in that locality. A DPE MUST reject handles that do not match the current locality. New handles are bound to the current locality except in two ways:

- The `target-locality` argument for the `DeriveContext` command is used to assign a different locality to the derived context. The returned `new-context-handle` will be bound to the locality specified by the `target-locality` argument.
- A context is explicitly moved to another locality by using the `RotateContextHandle` command and providing the `target-locality` argument.

The length and format of the locality identifier, `target-locality`, is specified by a DPE profile.

If a DPE supports multiple localities, it MUST support a distinct plaintext session for each locality. As a result, each locality also has a distinct default context. A DPE MAY simultaneously support localities that support encrypted sessions and localities that do not support encrypted sessions. For example, commands from normal applications might use encrypted sessions while commands from specialized hardware domains do not. When a context is moved from a locality that supports encrypted sessions to a locality that does not support encrypted sessions, the session binding is removed from the context. A context cannot be moved from a locality that does not support sessions to a locality that does.

Start of Informative Comment

Implementers may ignore locality if there is no way for their DPE to differentiate between clients apart from session IDs or context handles. In this case, the DPE would be considered a single-locality DPE.

Examples of existing technologies that might be used to implement localities include:

- CPU modes (e.g., kernel vs user mode) or protection rings
- ARM TrustZone secure vs non-secure world distinction
- Bus technology that differentiates multiple peripherals, e.g., Serial Peripheral Interface (SPI)

End of Informative Comment

5.10 Messages

This section provides requirements related to command and response messages that comprise the DPE message-based interface.

5.10.1 Transport

Apart from the encrypted session considerations, message transport is not constrained in any way by this specification: it is out of scope.

5.10.2 Encoding

All DPE command and response messages MUST be less than or equal to 65535 bytes in length including encryption overhead. A DPE implementation SHOULD support messages of at least 4096 bytes. Messages are encoded using a constrained subset of the RFC8949 CBOR format [9]. The following additional constraints exist to promote implementation simplicity and correctness. The additional constraints are as follows:

1) Deterministically encoded CBOR is REQUIRED, as specified in RFC8949 section 4.2.1 [9]
2) Floating point numbers and tags MUST NOT be used
3) Map keys other than integers MUST NOT be used

A DPE MUST follow these rules when generating messages and SHOULD enforce these rules on incoming messages by responding with an `invalid-command` or `invalid-argument` error.

Each command and response defines a CBOR map for arguments, with each field being optional. Optional fields allow for future extensibility at the encoding level, but this does not indicate the fields are optional semantically. Each command specifies whether an input argument field is required and, if not, a default value. Similarly, each response specifies when an output argument will be omitted. A future version of this specification could make a previously required argument optional as part of a deprecation process, for example.

Messages are described in this document using CDDL [10]. CDDL sockets are used for indicating where choice types are expected to be extended in future versions of the specification or with vendor-defined extensions.

5.10.3 Session Message Format

Each command or response message is associated with a session and is encoded as a CBOR array, as illustrated in the following CDDL snippet:

```
session-message = [
  session-id: uint,
  message: bytes,  ; Ciphertext, unless using the plaintext session.
]
```

For an encrypted session, the entire message is encrypted except for `session-id`. The message format and encoding are determined by the session protocol. When using the recommended Noise protocol, the message field is a Noise transport message, which is simply an AEAD ciphertext.

5.10.4 Command and Response Headers

For every command message, the input fields described for the command are appended to a common header that identifies the command. Response messages have a similar header with an error code. The format of command and response messages, where `input-args` and `output-args` are command-specific maps, is illustrated in the following CDDL snippet:

```
command-message = [
  command-id: $command-id,
  input-args: $input-args,
]

response-message = [
  error-code: $error-code,
  output-args: $output-args,
]

$command-id /= &(get-profile: 1)
$command-id /= &(open-session: 2)
$command-id /= &(close-session: 3)
$command-id /= &(sync-session: 4)
$command-id /= &(initialize-context: 7)
$command-id /= &(derive-context: 8)
$command-id /= &(certify-key: 9)
$command-id /= &(sign: 10)
$command-id /= &(seal: 11)
$command-id /= &(unseal: 12)
$command-id /= &(derive-sealing-public-key: 13)
$command-id /= &(rotate-context-handle: 14)
$command-id /= &(destroy-context: 15)
$command-id /= &(get-certificate-chain: 16)

$error-code /= &(no-error: 0)
$error-code /= &(internal-error: 1)
$error-code /= &(invalid-command: 2)
```

```
$error-code /= &(invalid-argument: 3)
$error-code /= &(session-exhausted: 4)
$error-code /= &(initialization-seed-locked: 5)
$error-code /= &(out-of-memory: 6)
$error-code /= &(cancelled-command: 7)

$input-args /= get-profile-input-args
$input-args /= open-session-input-args
$input-args /= close-session-input-args
$input-args /= sync-session-input-args
$input-args /= initialize-context-input-args
$input-args /= derive-context-input-args
$input-args /= get-certificate-chain-input-args
$input-args /= certify-key-input-args
$input-args /= sign-input-args
$input-args /= seal-input-args
$input-args /= unseal-input-args
$input-args /= derive-sealing-public-key-input-args
$input-args /= rotate-context-handle-input-args
$input-args /= destroy-context-input-args

$output-args /= get-profile-output-args
$output-args /= open-session-output-args
$output-args /= close-session-output-args
$output-args /= sync-session-output-args
$output-args /= initialize-context-output-args
$output-args /= derive-context-output-args
$output-args /= get-certificate-chain-output-args
$output-args /= certify-key-output-args
$output-args /= sign-output-args
$output-args /= seal-output-args
$output-args /= unseal-output-args
$output-args /= derive-sealing-public-key-output-args
$output-args /= rotate-context-handle-output-args
$output-args /= destroy-context-output-args
```

5.10.5 Multi Part Operations

If the transport between a DPE and client is constrained, some command or response arguments may not fit in a single transport message. While message chunking can be solved entirely at the transport layer, leaving the messages at the endpoints unaffected, this may be undesirable in some cases and a solution at the command layer is preferred. For this reason, a DPE can support multi-part messages. Whether a DPE supports multi-part messages is indicated in its DPE profile.

Multi-part messages are simply messages with additional arguments to facilitate the use of multiple command-response pairs for a single operation that would otherwise comprise a single command and single response. The behavioral requirements for all commands and responses are the same regardless of whether multi-part messages are used.

A multi-part message flow consists of a series of command-response pairs. Each command in the series has the same command ID, and each response has an error code indicating the status so far. The command-response pairs continue until all inputs and all outputs have been fully transferred.

A DPE that supports multi-part messages might also support concurrent operations. An opaque operation handle is used to resolve messages for concurrent operations. If a DPE does not support concurrent operations per its profile, the operation handle can be omitted. If a DPE supports operation handles, each handle MUST be generated by the DPE with the same security properties as required for a context handle, see section 5.6. When operation handles are used, a DPE SHALL assign a handle to each multi-part operation and MAY rotate an operation handle during multi-part operations. In each response during a multi-part operation, a DPE will provide the current operation handle to the client. The client will include that operation handle in the next command in the multi-part operation.

A DPE has the following options for use of operation handles:

- One operation handle can be used per operation. The same operation handle is used for each message in the multi-part operation. Or,
- A fresh operation handle can be used per response message, forcing a client to process the response containing the current operation handle before submitting the next command.

For multi-part operations a DPE SHALL provide a response for each part of the operation. When all client data has been received by the DPE, but the DPE has not provided all the output data, the client sends a command containing `more-data` and `operation-handle` arguments but no other input. The DPE will respond with a message containing the `more-data` argument, the current `operation-handle`, and the next chunk of output for the operation. Similarly, when not all input has been received by a DPE, the DPE will respond to a command message with a response that contains at least `more-data` and the current `operation-handle`.

Arguments in multi-part messages have the following constraints:

- Any argument of type bytes, except the `operation-handle` argument, can be split across messages as necessary. For any given message the argument field contains a single chunk of the entire argument value. Arguments of other types cannot be split and appear in any single message within the multi-part operation. If an argument of a type other than byte appears in more than one message within the same multi-part operation, a DPE MUST abort the operation with an `invalid-argument` error.
- The Boolean argument named `more-data` indicates whether more chunks are in any argument in a subsequent message. The default for `more-data` is false. If a DPE receives a command message with an input argument other than the `operation-handle` after it receives a command message with the `more-data` argument set to false, it MUST abort the operation with an `invalid-argument` error.
- If operation handles are used, command messages have an `operation-handle` argument of type bytes that contains the handle value returned by the most recent response message of the same operation. An `operation-handle` argument cannot be split.
- To avoid conflicts, the CBOR map key values for `more-data` and `operation-handle` are always 100 and 101 respectively, regardless of which command these are being added to.

A DPE MAY populate output arguments before input arguments have been fully received. For example, a DPE can use multi-part messages to stream both input and output for sealing or unsealing.

Chunks of a split argument are sent in order. In other words, a recipient can append chunks in the order they are received to form the full argument value. A DPE MUST interpret multiple chunks for an input argument as provided in order by the client. Similarly, a DPE MUST send output argument chunks to a client in order.

A DPE MUST allow a client to send input arguments and input argument chunks in any command message within the multi-part operation flow until the `more-data` input argument is set to false. For example, it is valid for a client to interleave chunks of different split input arguments or to provide a non-split argument between or alongside chunks of a split argument. A DPE implementation has similar flexibility with output arguments.

As an example, the following CDDL snippet demonstrates the addition of multi-part arguments to the `unseal-input-args` defined for the Unseal command in section 6.

```
unseal-input-args = {
  ? &(more-data: 100) => bool,
  ? &(operation-handle: 101) => bytes,
  ? &(context-handle: 1) => bytes,
  ? &(retain-context: 2) => bool,
  ? &(is-asymmetric: 3) => bool,
  ? &(label: 4) => bytes,
  ? &(data-to-unseal: 5) => bytes,       ; Can be split with more-data set to true
  * &(tstr: uint) => any
}
```

5.10.6 Reserved Command ID Values

This specification reserves command ID values 0 through 127 for future use. Implementers can use other ID values for custom commands.

5.11 Errors

A DPE MUST NOT become unresponsive as a result of input from a client. If a command cannot be completed successfully, for any reason, a DPE MUST respond with an error code.

If an internal or environmental condition precludes continued operation, a DPE MUST NOT resume operation until it is reset in a way that invalidates all internal state (e.g., sessions, handles, CDIs).

A DPE MUST NOT include any output arguments in an error response. DPE commands are considered transactional with respect to DPE state. In the event of an error, a DPE MUST remain in the state it was in before the failed command was received. This requirement does not apply to the transport or encrypted session-related state because the transport and/or encrypted session protocol(s) are independent of the DPE state and are specified by a DPE profile.

Transport errors are out of scope for this specification but are expected to be reported using another mechanism or, at least, error codes that differ from DPE errors. DPE errors are intended to be communicated to a client in response to a command. It is strongly recommended that implementations of transport layers do not mask or modify DPE error codes.

The following error codes are defined:

CODE	ERROR	DESCRIPTION
0	No error	Indicates no error has occurred
1	Internal Error	An unexpected error has occurred which is not actionable by the client
2	Invalid Command	The command could not be decrypted, parsed, or is not supported
3	Invalid Argument	A command argument is malformed, invalid with respect to the current DPE state, in conflict with other arguments, not allowed, not recognized, or otherwise not supported
4	Session Exhausted	Keys for an encrypted session have been exhausted
5	Initialization Seed Locked	The command cannot be fulfilled because an internal seed component is no longer available
6	Out of Memory	A lack of internal resources prevented the DPE from fulfilling the command as requested
7	Cancelled Command	The command was cancelled

Table 1: DPE error codes

5.12 Summary

This section summarizes DPE requirements related to contexts, sessions, and localities. This is not an exhaustive list of DPE requirements.

1) A DPE MUST support at least one locality and MAY support multiple localities.

Start of Informative Comment

DPEs that do not differentiate between localities are considered as supporting a single locality.

End of Informative Comment

2) A DPE MUST support at least one session.
3) A DPE MUST support exactly one plaintext session per locality.
4) A DPE SHOULD support (a) one or more encrypted sessions and/or, (b) multiple localities.
5) A DPE MUST ensure each context has exactly one valid context handle (or is a default context with no client-visible handle) and is bound to exactly one session and one locality at any given time.
6) For each session, regardless of type, a DPE MUST support either: (a) a single default context, with no client-visible handle, or (b) one or more contexts that are each allocated a context handle known to a client.
7) A DPE that supports encrypted sessions SHALL fail (by returning `invalid-command`) all commands sent via the plaintext session except for `OpenSession`, `GetProfile`, and `SyncSession`.

Start of Informative Comment

A DPE that supports encrypted sessions will use the plaintext session only to establish an encrypted session.

End of informative Comment

6 COMMANDS

This section describes DPE commands, including the format of arguments in command and response messages. A DPE MUST support the `DeriveContext` and `DestroyContext` commands, and at least one of `Sign` or `Unseal`. A DPE supports other commands according to its profile. If a client attempts to invoke an unsupported command, a DPE SHALL respond with the `invalid-command` error. If a client includes an unsupported argument, a DPE SHALL respond with the `invalid-argument` error. If a client omits a required argument or provides conflicting arguments as specified in the command descriptions in this specification, a DPE SHALL respond with the `invalid-argument` error.

6.1 GetProfile

This command queries a DPE's profile. Information about a profile is returned as a profile descriptor.

Input Arguments

- None

Output Arguments:

- **`profile-descriptor`**: A CBOR-encoded description of the profile. See Section 7.4 for details on the format and semantics of the descriptor.

Argument Format:

```
get-profile-input-args = {* &(tstr: uint) => any}


get-profile-output-args = {
  ? &(profile-descriptor: 1) => profile-descriptor,
  * &(tstr: uint) => any
}
```

6.2 OpenSession

This command establishes a new encrypted session. The initiator and responder messages are formatted according to the session protocol and MAY be unencrypted or partially encrypted. The responder message of the session protocol MUST contain the new session ID encoded as a CBOR uint. Protocols that require more than a single round trip for session establishment are not supported by this command. When using the recommended session protocol, each handshake message contains a payload field: the initiator payload is empty, and the responder payload contains the session ID.

Input Arguments

- **`initiator-handshake`**: A handshake message from the initiator to responder. The format and semantics are determined by the session protocol. This argument is REQUIRED.

Output Arguments

- **`responder-handshake`**: A handshake message from responder to initiator: the format and semantics are determined by the session protocol. This message contains the new session ID as a payload.

Argument Format

```
open-session-input-args = {
  ? &(initiator-handshake: 1) => bytes,
  * &(tstr: uint) => any
}

open-session-output-args = {
  ? &(responder-handshake: 1) => bytes,
  * &(tstr: uint) => any
}

responder-handshake-payload = uint  ; The new session ID
```

6.3 CloseSession

This command closes a session. All context handles and data that are bound to the session, regardless of type, are destroyed and the session ID is invalidated. The session that is closed is the session used to send the command. As a result, only a client of a given session can close that session. A DPE can reuse session IDs, so knowing or guessing a session ID is insufficient to close the session.

When a plaintext session is closed, all contexts bound to it are destroyed as with any other session, but the plaintext session will always remain valid. When a DPE supports encrypted sessions, calling `CloseSession` on the plaintext session is not meaningful, since no contexts can be bound to it.

Input Arguments

- None

Output Arguments

- None

Argument Format

```
close-session-input-args = {* &(tstr: uint) => any}

close-session-output-args = {* &(tstr: uint) => any}
```

6.4 SyncSession

This command synchronizes an encrypted session and MUST be invoked using the plaintext session. This command is useful for some session protocols if undelivered messages cause the client and DPE to fall out of sync. The responder (DPE) updates its copy of the initiator (client) counter, and the initiator updates its copy of the responder counter. In both cases, the counter is updated if and only if the new counter value is larger than the current value.

If a session protocol does not require message synchronization, or does not do so using counters, this command has no effect.

Input Arguments

- **`session-id`**: The session ID of the session to be synchronized. This argument is REQUIRED.

- **`initiator-counter`**: The initiator's copy of the session counter for messages originating from the initiator. If this value is larger than the DPE's copy of `initiator-counter`, the DPE updates its copy. The DPE can enforce other requirements on the counter's value according to the session protocol. If omitted, the default value is zero.

Output Arguments

- **`responder-counter`**: The DPE's copy of the session counter for messages originating from the DPE. The client updates its copy of this counter if the new value is larger than the client's copy of `responder-counter` and meets other requirements per the session protocol.

Argument Format

```
sync-session-input-args = {
  ? &(session-id: 1) => uint,
  ? &(initiator-counter: 2) => uint,
  * &(tstr: uint) => any
}

sync-session-output-args = {
  ? &(responder-counter: 1) => uint,
  * &(tstr: uint) => any
}
```

6.5 InitializeContext

This command initializes a new DPE context. See section 5.6 for details on DPE contexts.

Input Arguments

- **`simulation`**: Indicates whether to create a simulation context. If omitted, the default is false.
- **`use-default-context`**: Indicates whether to use the default context for the current session instead of returning a context handle to the client. If omitted, the default is false.
- **`seed`**: This argument provides a seed value as an input to the initialization. A DPE profile specifies how this seed is used and specifies requirements for secure operation, if any. For example, the seed might be used as a UDS. This argument may be required by a DPE profile.

Output Arguments

- **`new-context-handle`**: A context handle for the new context. Omitted if the default context was used.

Argument Format

```
initialize-context-input-args = {
  ? &(simulation: 1) => bool,  ; Default = false
  ? &(use-default-context: 2) => bool,  ; Default = false
  ? &(seed: 3) => bytes,
  * &(tstr: uint) => any
}


initialize-context-output-args = {
  ? &(new-context-handle: 1) => bytes,
  * &(tstr: uint) => any
}
```

6.6 DeriveContext

This command performs the DICE computation [1] on a given set of inputs. The `DeriveContext` command is the fundamental DICE operation. It is used to derive the CDI value for a child component given a set of inputs that describe the component. Other DPE commands are either made possible by this command or exist to make this command possible.

Many details of how this command behaves are specified by a DPE profile, including:

- The format of input data
- The mapping of input data to the derivation computations
- The mapping of input data to certificate fields
- The availability and semantics of internal inputs
- The algorithm to derive new CDIs, asymmetric keys, and other certificate data
- The format of the new certificate

Input Arguments

- **`context-handle`**: A context handle for the client's current DPE context, see Section 5.6. This can be a simulation context. If derivation is not allowed for this context (see the `allow-new-context-to-derive` argument) a DPE MUST abort this operation and return an `invalid-argument` error. If omitted, the default context is used. If `retain-parent-context` is true, this context will be retained in its current state for subsequent operations and a new context handle for this context will be returned to the client. This is useful if a parent program creates multiple child programs and computes CDI(s) for each. If the default context is used and `retain-parent-context` is true, the derived context MUST use a different default context and the following apply:
    - At least one of `new-session-initiator-handshake` or `target-locality` is REQUIRED.
    - If a new session is created, the derived context MUST become the default context of the new session.
    - If a new session is not created, the `target-locality` argument MUST indicate a locality other than the current locality and the derived context MUST become the default context of the target locality's plaintext session.
    - The parent context is retained as the current session's default context.
- **`retain-parent-context`**: Indicates whether the parent context is to be retained, as explained in the description of the `context-handle` input argument. If omitted, the default value is false.
- **`allow-new-context-to-derive`**: Indicates whether the derived context is allowed to be used in a subsequent invocation of `DeriveContext`. This is useful if the derived context is known to belong to a program that should not derive additional DPE contexts, for example, an application. The `allow-new-context-to-derive` argument is similar in meaning to, for example, a CA specifying pathLen=0 in X.509v3 basicConstraints. If a DPE supports X.509 certificates, the DPE SHOULD set pathLen=0 in X.509v3 basicConstraints within the corresponding certificate when this argument is false. If omitted, the default value is true.

- **`create-certificate`**: Indicates whether to create an intermediate certificate for this component, which can be an ECA certificate as defined by DICE Certificate Profiles. If omitted, the default is true. If this argument is set to false, no certificate is generated for the component and any information that would normally have been added to the certificate is accumulated as part of the context and will appear in the next certificate generated, whether by a subsequent `DeriveContext` command or a `CertifyKey` command. The private key corresponding to the most recent certificate generated MUST be retained, even if `retain-parent-context` is set to false. A DPE MUST NOT permit the accumulated certificate information to be removed or modified until it is represented in a certificate via a subsequent invocation of this command.
- **`new-session-initiator-handshake`**: This argument is used to create a new session for the derived context. This is the initiator handshake message for the new session and the corresponding handshake response is returned as an output argument. Session protocols can use binding information from the current session to negotiate the new session, but the resulting new session MUST be independent of the current session. If omitted, the derived context will be bound to the current session. See section 5.7.1.
- **`input-data`**: Input from a DICE component (i.e., client) that describes all security-relevant properties of the child component. How this data is formatted, used in the DICE computation, mapped to certificate fields, etc. is determined by a DPE profile. This value can be a TCB Component Identifier (TCI); see [2]. This argument is REQUIRED: there is no default value.
- **`internal-inputs`**: An array of references to internal inputs to include in the DICE computation. This argument does not contain the actual input values. If omitted, no internal inputs are used. This argument indicates which inputs should be included in the computation: this argument does not contain internal inputs. The inputs indicated by this argument are held internal to the DPE, and their availability and behavior is governed by a DPE profile. A profile can define arbitrary internal inputs in addition to those defined here. Uses include anything that needs protection or policy enforcement (e.g., monotonic counters, rotatable secrets, etc.). Internal inputs indicated in this argument MUST be included in the DICE computation and can be included in a certificate. The following internal inputs are defined:
    - **`dpe-info`**: This contains basic information about the DPE: version, configuration, profile, etc.
    - **`dpe-dice`**: This contains internal DICE state of the DPE, including CDI(s), certificate chain, etc.
- **`target-locality`**: Identifies the locality to which the derived context will be bound. If omitted, the derived context will be bound to the current locality. If the target locality does not support encrypted sessions, the derived context will be bound to the plaintext session of the target locality. If the target locality supports encrypted sessions, the DPE MUST ensure the current locality also supports encrypted sessions and that the derived context session binding is unaffected by the locality change. Otherwise, the DPE MUST return `invalid-argument`. Except for requirements outlined in the `context-handle` argument description, this argument is OPTIONAL.
- **`return-certificate`**: Indicates whether a DPE MUST return the generated certificate when `create-certificate` is true. If true, the certificate is returned in the `certificate` output argument. If `create-certificate` is false, this argument is ignored. If omitted, the default value of this argument is false.
- **`allow-new-context-to-export`**: Indicates whether the DPE permits export of the CDI from the newly derived context. If false, a DPE MUST ensure subsequent `DeriveContext` operations using the derived context do not allow `export-cdi` to be set to true. Once disabled, this attribute is inherited and cannot be re-enabled on subsequent derivations from this context forward. A DPE MUST NOT allow export to be enabled for a derived context when the parent context does not have export enabled. If omitted, the default value is false.
- **`export-cdi`**: Whether to export the derived CDI. The format and encoding of the exported CDI are specified by a DPE profile. If true, a DPE MUST include an indication that the CDI is to be exported in the DICE computation and certificate, if applicable. A DPE profile defines this indicator in the form of an internal input. A DPE MUST NOT proceed with export and return an `invalid-argument` error when export is not allowed for the parent context. Once a CDI is exported, the DPE MUST NOT retain the derived context and the DPE MUST omit the `new-context-handle` output argument. If a DPE supports certificates, the DPE MUST require

that the `create-certificate` argument is true and return an `invalid-argument` error if it is false. The CDI cannot be exported from a simulation context and a DPE MUST return an `invalid-argument` error if `context-handle` refers to a simulation context. A DPE does not control the CDI upon successful completion of this command. When this argument is true, a DPE MUST return an `invalid-argument` error if either `allow-new-context-to-derive` or `allow-new-context-to-export` are false, or if a `new-session-initiator-handshake` or `target-locality` is supplied. If this argument is omitted, the default is false.

- **`recursive`**: Indicates whether the derivation should recursively affect all contexts previously derived from the given context in addition to affecting the given context. If omitted, the default value is false. If true, the changes to each affected context MUST appropriately reflect the given `input-data`. A DPE profile defines whether recursive derivation is supported and, if so, how contexts are affected and the way the change is applied to each context. Recursive derivation is a feature that enables on-the-fly update of system components. It is expected to be infrequent. It can be disruptive to the clients that use the recursively affected contexts since state related to key derivation and unseal policy might change. For each affected context, the handle value is unchanged and the parent context relation is unchanged (except that the parent is also affected by this operation). The parent context cannot be retained for a recursive derivation, so if the `recursive` argument is true then a DPE MUST return an `invalid-argument` error if `retain-parent-context` is set to true. Similarly, a recursive derivation cannot export a CDI or return a certificate so, if the `recursive` argument is true then a DPE MUST return an `invalid-argument` error if `export-cdi` or `return-certificate` is set to true. A DPE MUST apply all other arguments recursively. For example, if `new-session-initiator-handshake` is supplied, all affected contexts will be bound to the new session.

Output Arguments

- **`new-context-handle`**: A context handle for the derived context. This will be omitted if the default context is used or if a CDI is exported.
- **`new-session-responder-handshake`**: If a new session was initiated by including the `new-session-initiator-handshake` input argument, this is the corresponding handshake message from the protocol responder. The new session is already fully operational on the DPE side, and the derived context is associated with the new session.
- **`new-parent-context-handle`**: If the parent context was retained and the default context was not used, this argument contains a new context handle for the parent context.
- **`new-certificate`**: If `create-certificate` and `return-certificate` are both true, this argument is the new certificate generated for the new context.
- **`exported-cdi`**: If `export-cdi` is true, this argument is the exported CDI value.

Argument Format

```
derive-context-input-args = {
  ? &(context-handle: 1) => bytes,
  ? &(retain-parent-context: 2) => bool,  ; Default = false
  ? &(allow-new-context-to-derive: 3) => bool,  ; Default = true
  ? &(create-certificate: 4) => bool,  ; Default = true
  ? &(new-session-initiator-handshake: 5) => bytes,
  ? &(input-data: 6) => bytes,
  ? &(internal-inputs: 7) => [* $internal-input-type],
  ? &(target-locality: 8) => bytes,
  ? &(return-certificate: 9) => bool,  ; Default = false
  ? &(allow-new-context-to-export: 10) => bool,  ; Default = false
  ? &(export-cdi: 11) => bool,  ; Default = false
  ? &(recursive: 12) => bool,  ; Default = false
  * &(tstr: uint) => any
}

$internal-input-type /= &(
  dpe-info: 1,
  dpe-dice: 2,
)

derive-context-output-args = {
  ? &(new-context-handle: 1) => bytes,
  ? &(new-session-responder-handshake: 2) => bytes,
  ? &(parent-context-handle: 3) => bytes,
  ? &(new-certificate: 4) => bytes,
  ? &(exported-cdi: 5) => bytes,
  * &(tstr: uint) => any
}
```

6.7 GetCertificateChain

This command returns the certificate chain generated for a given DPE context. The order, format, and encoding of the certificate chain are specified by a DPE profile. If the context contains accumulated certificate information that is not yet part of a certificate, a DPE MUST abort the operation and return an `invalid-argument` error. The connection of the certificate chain to external infrastructure like a manufacturer-issued certificate is also defined by a profile. A profile can specify that additional certificates not generated by the DPE are included in the certificate chain returned by this command. It is possible for a certificate chain to be empty, and it is valid for this command to succeed and return no certificates.

Input Arguments

- **`context-handle`**: A handle for the context from which to retrieve the certificate chain. If omitted, the default context is used. The context MAY be a simulation context.
- **`retain-context`**: Indicates whether the DPE context is to be retained for subsequent commands. If false, the context is destroyed (see `DestroyContext`). If true, a new context handle will be returned to the client as an output argument unless the default context is used. If the default context is used and this argument is true, the context will be retained as default. If omitted, the default is false.
- **`clear-from-context`**: Whether a DPE MUST clear the certificate chain from the context so subsequent `GetCertificateChain` operations on the given context, or contexts derived from it, do not include the certificates returned by this command. If `retain-context` is false, this argument is ignored. If omitted, the default is false.

Output Arguments

- **`certificate-chain`**: The certificate chain. The content and format of the certificates depend on the DPE profile.
- **`new-context-handle`**: A new handle for the DPE context if the `retain-context` argument was set to true.

Argument Format

```
get-certificate-chain-input-args = {
  ? &(context-handle: 1) => bytes,
  ? &(retain-context: 2) => bool,  ; Default = false
  ? &(clear-from-context: 3) => bool,  ; Default = false
  * &(tstr: uint) => any
}

get-certificate-chain-output-args = {
  ? &(certificate-chain: 1) => [* bytes],
  ? &(new-context-handle: 2) => bytes,
  * &(tstr: uint) => any
}
```

6.8 CertifyKey

This command certifies a signing key using the given DPE context as the certification authority. If the public key to certify is not provided as an input argument, a key pair is deterministically derived from the context and the label argument. If the public key to certify is not provided, a DPE MUST use the same method for deriving the key as it uses for the `Sign` command. This means the key derived by a DPE for `CerifyKey` will be the same key derived by the DPE for a `Sign` command using the same label. The new public key is certified and returned separate from the certificate in the response. If the context contains accumulated certificate information, that information MUST be represented in the new leaf certificate. The accumulated certificate information MUST remain in the context unmodified.

The content and format of certificates generated by the DPE and the type of asymmetric keys supported are specified by a DPE profile. It is recommended that the leaf certificate follows the requirements specified by DICE Certificate Profiles [4], and the policies argument can be used to indicate which policies to apply to the new leaf certificate.

Input Arguments

- **`context-handle`**: A handle for the context that will be used to issue a certificate for the signing key. If omitted, the default context is used. A DPE MUST NOT allow a simulation context to be used when the `public-key` argument is provided by the client, because the corresponding private key is not controlled by the DPE.
- **`retain-context`**: Indicates whether the DPE context is to be retained for subsequent commands. If false, the context is destroyed (see `DestroyContext`). If true, a new context handle will be returned to the client as an output argument unless the default context is used. If the default context is used and the `retain-context` argument is true, the context will be retained as default. If omitted, the default is false.
- **`public-key`**: The public key to certify. The type and format of the public key is specified by a DPE profile. If omitted, a key pair is deterministically derived from the context and the label argument.
- **`label`**: A label to use as additional input to the asymmetric key derivation from the context. If `public-key` is provided, there is no derivation, and this argument is ignored. Using the same label with the same context

multiple times will yield the same key pair. If omitted, an empty label is used (i.e., a label of zero length). A DPE profile may define a fixed set of supported labels.

- **`policies`**: A set of policy values that determine the construction of the certificate. When a DPE profile uses the policies defined by DICE Certificate Profiles [4] this argument specifies which of the policies should apply to the new leaf certificate. A DPE profile determines which of these policies are supported, if any, and may define other policies. A DPE profile determines the behavior when this argument is omitted.
- **`additional-input`**: Additional input to be used when generating the leaf certificate. The format and semantics of this input data are specified by a DPE profile. This argument is OPTIONAL.

Output Arguments

- **`certificate`**: The new leaf certificate. The content and format of the certificate depends on the DPE profile.
- **`derived-public-key`**: The public key of the derived key pair, if any. The type and format of the public key is specified by a DPE profile. If the `public-key` input argument was provided, there is no derived key pair, and this argument is omitted.
- **`new-context-handle`**: A new handle for the DPE context if the `retain-context` argument was set to true.

Argument Format

```
certify-key-input-args = {
  ? &(context-handle: 1) => bytes,
  ? &(retain-context: 2) => bool,  ; Default = false
  ? &(public-key: 3) => bytes,
  ? &(label: 4) => bytes,
  ? &(policies: 5) => [* $policy-type],
  ? &(additional-input: 6) => bytes,
  * &(tstr: uint) => any
}

$policy-type /= &(
  tcg-dice-kp-identityInit: 6,  ; Matches the corresponding OID
  tcg-dice-kp-identityLoc: 7,
  tcg-dice-kp-attestInit: 8,
  tcg-dice-kp-attestLoc: 9,
  tcg-dice-kp-assertInit: 10,
  tcg-dice-kp-assertLoc: 11,
)

certify-key-output-args = {
  ? &(certificate: 1) => bytes,
  ? &(derived-public-key: 2) => bytes,
  ? &(new-context-handle: 3) => bytes,
  * &(tstr: uint) => any
}
```

6.9 Sign

This command signs a given message with a key derived from the given DPE context and label. The signature algorithm is specified by a DPE profile. A DPE profile may support an asymmetric signature scheme and/or a symmetric signature scheme (e.g., a MAC). For asymmetric signatures, the signing key will be the same key derived by `CertifyKey` for the same label. The DPE context MUST NOT be a simulation context.

Input Arguments

- **`context-handle`**: A handle for the DPE context that will be used to derive a signing key. If omitted, the default context is used. A DPE MUST NOT allow a simulation context to be used.
- **`retain-context`**: Indicates whether the DPE context is to be retained for subsequent commands. If false, the context is destroyed (see `DestroyContext`). If true, a new context handle will be returned to the client as an output argument unless the default context is used. If the default context is used and the `retain-context` argument is true, the context will be retained as default. If omitted, the default is false.
- **`label`**: A label to use as additional input to the asymmetric key derivation from the DPE context. Using the same label with the same DPE context multiple times will yield the same key pair. If omitted, an empty label is used (i.e., a `label` of zero length). A DPE profile may define a fixed set of supported labels.
- **`is-symmetric`**: Indicates whether a symmetric signature scheme should be used. If omitted, the default value is false.
- **`to-be-signed`**: The data to be signed. The format of this data is specified by a DPE profile. This argument is REQUIRED.

Output Arguments

- **`signature`**: The signature over the data in the `to-be-signed` input argument. The format of the signature is specified by a DPE profile.
- **`new-context-handle`**: A new handle for the DPE context if the `retain-context` argument was set to true.

Argument Format

```
sign-input-args = {
  ? &(context-handle: 1) => bytes,
  ? &(retain-context: 2) => bool,  ; Default = false
  ? &(label: 3) => bytes,
  ? &(is-symmetric: 4) => bool,  ; Default = false
  ? &(to-be-signed: 5) => bytes,
  * &(tstr: uint) => any
}


sign-output-args = {
  ? &(signature: 1) => bytes,
  ? &(new-context-handle: 2) => bytes,
  * &(tstr: uint) => any
}
```

6.10 Seal

This command seals data to a given policy using a symmetric key deterministically derived from the given DPE context. The sealed data can be unsealed only by calling the `Unseal` command with the same DPE context and `is-asymmetric` set to false.

Start of Informative Comment

This command is not used for asymmetric sealing use cases. Asymmetric sealing allows a client to seal data without the DPE by using a public key (see `DeriveSealingPublicKey`).

End of Informative Comment

Input Arguments

- **`context-handle`**: A handle for the DPE context that will be used to derive a sealing key. This can be a simulation context. If omitted, the default context is used.
- **`retain-context`**: Indicates whether the DPE context is to be retained for subsequent commands. If false, the context is destroyed (see `DestroyContext`). If true, a new context handle will be returned to the client as an output argument unless the default context is used. If the default context is used and the `retain-context` argument is true, the context will be retained as default. If omitted, the default is false.
- **`unseal-policy`**: This argument describes a policy that constrains the conditions under which the DPE will allow the `sealed-data` output argument to be unsealed. The format and semantics of this policy value are specified by a DPE profile. A DPE MUST include this value in the sealing key derivation to ensure the same policy is required for a successful unseal. This argument may be required by a DPE profile.
- **`label`**: A label to be included in the sealing key derivation. If omitted, an empty label is used (i.e., a label of zero length). A DPE profile may define a fixed set of supported labels.
- **`data-to-seal`**: The data to be sealed. A DPE profile can place constraints on the length, content, and format of this data. This argument is REQUIRED.

Output Arguments

- **`sealed-data`**: The sealed data as opaque bytes. The format and content are implementation dependent.
- **`new-context-handle`**: A new handle for the DPE context if the `retain-context` argument was set to true.

Argument Format

```
seal-input-args = {
  ? &(context-handle: 1) => bytes,
  ? &(retain-context: 2) => bool,  ; Default = false
  ? &(unseal-policy: 3) => bytes,
  ? &(label: 4) => bytes,
  ? &(data-to-seal: 5) => bytes,
  * &(tstr: uint) => any
}


seal-output-args = {
  ? &(sealed-data: 1) => bytes,
  ? &(new-context-handle: 2) => bytes,
  * &(tstr: uint) => any
}
```

6.11 Unseal

This command unseals data previously sealed to a given policy with a symmetric or asymmetric key derived from the given DPE context. With a symmetric key, data previously sealed using the `Seal` command can be unsealed. With an asymmetric key, data previously sealed using the public key produced by the `DeriveSealingPublicKey` command as specified by a DPE profile can be unsealed.

Input Arguments

- **`context-handle`**: A handle for the DPE context that will be used to derive a sealing key. If omitted, the default context is used. A DPE MUST NOT allow a simulation context to be used.
- **`retain-context`**: Indicates whether the DPE context is to be retained for subsequent commands. If false, the context is destroyed (see `DestroyContext`). If true, a new context handle will be returned to the client as an output argument unless the default context is used. If the default context is used and the `retain-context` argument is true, the context will be retained as default. If omitted, the default is false.
- **`is-asymmetric`**: Indicates whether to use a symmetric or asymmetric key to unseal. If omitted, the default value is false.
- **`unseal-policy`**: This argument describes a policy that constrains the conditions under which the DPE will allow the `data-to-unseal` output argument to be unsealed. The format and semantics of this policy value are specified by a DPE profile. A DPE MUST NOT proceed with an unseal operation if the conditions of this policy are not met, or if this policy does not match the policy value provided prior to sealing. For symmetric sealing, the DPE MUST ensure this value matches the `unseal-policy` value provided to the `Seal` command that produced the `data-to-unseal` argument value. For asymmetric sealing, the DPE MUST ensure this value matches the `unseal-policy` value provided to the `DeriveSealingPublicKey` command that produced the public key used to seal the `data-to-unseal` argument value. This argument is REQUIRED.
- **`label`**: A label to be included in the sealing key derivation. If omitted, an empty label is used (i.e., a label of zero length). A DPE profile may define a fixed set of supported labels.
- **`data-to-unseal`**: The data to be unsealed. This argument is REQUIRED. For symmetric unseal, this is the `sealed-data` returned by the `Seal` command. For asymmetric unseal, the seal algorithm and format of this data are specified by a DPE profile.

Output Arguments

- **`unsealed-data`**: The original unsealed data. For symmetric unseal, this is the same data as the `data-to-seal` argument for the `Seal` command.
- **`new-context-handle`**: A new handle for the DPE context if the `retain-context` argument was set to true.

Argument Format

```
unseal-input-args = {
  ? &(context-handle: 1) => bytes,
  ? &(retain-context: 2) => bool,  ; Default = false
  ? &(is-asymmetric: 3) => bool,  ; Default = false
  ? &(label: 4) => bytes,
  ? &(data-to-unseal: 5) => bytes,
  * &(tstr: uint) => any
}


unseal-output-args = {
  ? &(unsealed-data: 1) => bytes,
  ? &(new-context-handle: 2) => bytes,
  * &(tstr: uint) => any
}
```

6.12 DeriveSealingPublicKey

This command provides a sealing public key derived from the given DPE context for a given policy and a label. Data sealed with this key can only be unsealed by calling the `Unseal` command with the same context and `is-asymmetric` set to true. A DPE MUST derive a different asymmetric key pair for sealing than it does for signing in `CertifyKey` and `Sign`, even if the CDI is the same.

Input Arguments

- **`context-handle`**: A handle for the DPE context that will be used to derive a sealing key. This can be a simulation context. If omitted, the default context is used.
- **`retain-context`**: Indicates whether the DPE context is to be retained for subsequent commands. If false, the context is destroyed (see `DestroyContext`). If true, a new context handle will be returned to the client as an output argument unless the default context is used. If the default context is used and the `retain-context` argument is true, the context will be retained as default. If omitted, the default is false.
- **`unseal-policy`**: This argument describes a policy that constrains the conditions under which the DPE will allow data sealed with the `derived-public-key` output argument to be unsealed. The format and semantics of this policy value are specified by a DPE profile. A DPE MUST include this value in the sealing key derivation to ensure the same policy is required for a successful unseal. This argument is REQUIRED.
- **`label`**: A label to be included in the sealing key derivation. If omitted, an empty label is used (i.e., a label of zero length). A DPE profile may define a fixed set of supported labels.

Output Arguments

- **`derived-public-key`**: The public key of the asymmetric key pair for sealing. The type and format of the public key are specified by a DPE profile. How this public key can be used to seal data is also specified by a DPE profile.
- **`new-context-handle`**: A new handle for the DPE context if the `retain-context` argument was set to true.

Argument Format

```
derive-sealing-public-key-input-args = {
  ? &(context-handle: 1) => bytes,
  ? &(retain-context: 2) => bool,  ; Default = false
  ? &(unseal-policy: 3) => bytes,
  ? &(label: 4) => bytes,
  * &(tstr: uint) => any
}


derive-sealing-public-key-output-args = {
  ? &(derived-public-key: 1) => bytes,
  ? &(new-context-handle: 2) => bytes,
  * &(tstr: uint) => any
}
```

6.13 RotateContextHandle

This command rotates a DPE context handle. The current handle is invalidated, and a new handle is returned. The context itself is unaffected.

When operating upon a default context, this command assigns a new handle to the context so it is no longer the default context. After this, the default context is invalid (unusable) until it is initialized again and there are no context handles in the current session.

Input Arguments

- **`context-handle`**: The handle to invalidate. If omitted, the default context is used.
- **`to-default`**: Indicates whether the context should become the default context of its session. If true, the default context MUST NOT already be valid, and no new handle is returned. If false, a new handle is returned for the context. A DPE MUST return `invalid-argument` if a caller attempts to rotate a default context to itself. If omitted, the default is false.
- **`target-locality`**: Identifies the locality to which the context will be bound. If omitted, the context will be bound to the current locality. If the target locality does not support encrypted sessions, the context will be bound to the plaintext session of the target locality. If the target locality supports encrypted sessions, the DPE MUST ensure the current locality also supports encrypted sessions and that the context session binding is unaffected by the locality change. Otherwise, the DPE MUST return `invalid-argument`. This argument is OPTIONAL.

Output Arguments

- **`new-context-handle`**: A new handle for the context. This argument is omitted on successful completion if `to-default` was true.

Argument Format

```
rotate-context-handle-input-args = {
  ? &(context-handle: 1) => bytes,
  ? &(to-default: 2) => bool,  ; Default = false
  ? &(target-locality: 3) => bytes,
  * &(tstr: uint) => any
}


rotate-context-handle-output-args = {
  ? &(new-context-handle: 1) => bytes,
  * &(tstr: uint) => any
}
```

6.14 DestroyContext

This command destroys a DPE context. After this command succeeds, the context handle is no longer usable. If the default context is destroyed, it becomes invalid (unusable) until it is initialized again. If the `destroy-recursively` argument is true, a DPE MUST destroy not only the indicated context but all contexts that have been derived from the context, recursively.

Input Arguments

- **`context-handle`**: A handle for the context to be destroyed. If omitted, the default context is used.
- **`destroy-recursively`**: Indicates whether all derived contexts should also be destroyed, recursively. If omitted, the default value is false.

Output Arguments

- None

Argument Format

```
destroy-context-input-args = {
  ? &(context-handle: 1) => bytes,
  ? &(destroy-recursively: 2) => bool,  ; Default = false
  * &(tstr: uint) => any
}

destroy-context-output-args = {* &(tstr: uint) => any}
```

7 PROFILES

A DPE profile specifies details where a DPE has flexibility. A profile is *complete* in that if a DPE design decision impacts interoperability between a client and a DPE, it MUST be specified by a profile. A profile includes attributes such as which features are supported or not supported, limits such as maximum message size, and formats such as inputs or public keys.

To define a profile, simply specify every attribute. This specification defines the attributes and provides a sample value for each attribute. See sections 7.2 and 7.3.

7.1 Namespaces

Names identify profiles and profile attribute values. Names are represented as a text string, a sequence of Unicode code points. Anyone can define a custom profile including various custom names without any kind of registration for those names. Names MUST be prefixed with an appropriate namespace to avoid collisions. The namespace SHOULD align with an internet domain owned by the defining entity. Namespaces prefixed with `tcg` are reserved for use by TCG and MUST NOT be used for custom names.

7.2 Profile Attributes

Table 2 describes each profile attribute. Each attribute is denoted with a tag-style name that carries over to the CDDL definition of a profile descriptor. Each attribute also has an expected type: Boolean, Number, or String. When attributes are encoded as a descriptor, these types are mapped to the CBOR types bool, uint, and tstr respectively. Custom string values MUST follow namespace requirements. Each custom name MUST reference one specific immutable value for the attribute, however complex, which means names are unambiguous. When defining the value associated with a name, any definition that meets the requirements of the attribute can be specified. Number fields can use the value ‘`Unlimited`’ to denote that no explicit constraint is imposed. String fields can use the value ‘`Empty`’ to denote the empty string.

In some cases, the value of one attribute renders another attribute irrelevant. For example, if encrypted sessions are not supported, defining a session protocol is not meaningful. In these cases, profiles SHOULD use the value ‘`NA`’ to indicate an attribute is not applicable and SHOULD omit the irrelevant attributes from the descriptor. In some cases, the value of one attribute places constraints on the value of another attribute. Both irrelevance and constraint implications are indicated in Table 2 with an **Implications** section describing anything that a value for the current attribute implies about other attributes. For convenience, an **Affected By** section lists attributes with implications that affect the current attribute. These documented implications preclude profiles that are inconsistently defined, which also means that self-consistency of a profile can be programmatically verified using these rules.

Some attributes are logically equivalent, like `supports-signing` and `supports-sign`, but they are still intentionally included for semantic clarity since the relationship is not always obvious.

Attribute	Type	Description
General		
`name`	String	The name of the current profile. A profile can be defined without a name: this is indicated with the empty string. When a name is provided (i.e., it is not the empty string), it MUST reference one specific immutable profile. In other words, a non-empty profile name is unambiguous in terms of the associated set of attribute values.

`inherits`	String	The name of a profile to inherit. When this value is not empty, all attributes omitted from the current profile use the value from this inherited profile. Any value specified for the current profile overrides inherited values. When this value is encoded in a descriptor, an encoded descriptor of the inherited profile is included instead of just the name of the inherited profile.
`dpe-spec-version`	Number	The supported DPE specification version. Only the major version is used since minor versions and revisions are mutually compatible.
`max-message-size`	Number	The maximum message size allowed at the transport layer. Note that this applies to command and response messages when using a message-based interface. When using a direct interface, this value is not meaningful.
`uses-multi-part-messages`	Boolean	Indicates whether a DPE uses multi-part messages for commands where these are defined. **Implications** ● If false, `supports-concurrent-operations` is irrelevant
`supports-concurrent-operations`	Boolean	Indicates whether a DPE supports concurrent multi-part command sequences. **Affected By** ● `uses-multi-part-messages`
Sessions		
`supports-encrypted-sessions`	Boolean	Indicates whether a DPE supports encrypted sessions. Note this also implies whether multiple sessions are supported, since there is only one plain-text session (session ID zero). **Implications** • When false, `supports-derived-sessions` and `supports-session-sync` MUST be false • When false, the following attributes are irrelevant and can be omitted: o `max-sessions` o `session-protocol` o `session-sync-policy` • When true, `supports-open-session` and `supports-close-session` MUST be true • When false, `supports-open-session`, `supports-close-session`, and `supports-sync-session` MUST be false

`supports-derived-sessions`	Boolean	Indicates whether a DPE supports the creation of new sessions via the `DeriveContext` command. **Affected By** - `supports-encrypted-sessions`
`max-sessions`	Number	The maximum number of open sessions supported. **Affected By** - `supports-encrypted-sessions`
`session-protocol`	String	Names the session protocol for encrypted sessions. The protocol MUST define these elements: - The content and format of handshake messages, including the session ID payload on response - The content and format of transport messages - The content and format of handshake messages for derived sessions via the `DeriveContext` command - The maximum number of ciphertext messages that can be exchanged per session, or a rotation scheme to handle key exhaustion if applicable **Affected By** - `supports-encrypted-sessions`
`supports-session-sync`	Boolean	Indicates whether a DPE supports session synchronization via the `SyncSession` command. **Implications** - When false, `session-sync-policy` is irrelevant and can be omitted **Affected By** - `supports-encrypted-sessions`
`session-sync-policy`	String	Names a policy enforced on the counters when synchronizing a session. The policy MUST define all checks required before accepting the new counter value. **Affected By** - `supports-encrypted-sessions` - `supports-session-sync`
Contexts		
`supports-default-context`	Boolean	Indicates whether a DPE supports default contexts. **Implications** - At least one of `supports-default-context` or `supports-context-handles` MUST be true - If false, `supports-auto-init` MUST be false

		Affected By - `supports-context-handles` - `supports-initialize-context`
`supports-context-handles`	Boolean	Indicates whether a DPE supports context handles, as opposed to only default contexts. **Implications** - At least one of `supports-context-handles` or `supports-default-context` MUST be true - If false, `supports-simulation` MUST be false - If false, `max-context-handle-size` is irrelevant and can be omitted **Affected By** - `supports-default-context`
`max-contexts-per-session`	Number	How many contexts can be associated with a single session.
`max-context-handle-size`	Number	The maximum size of context handles produced by a DPE. This is always subject to the `max-message-size` in a particular message. **Affected By** - `supports-context-handles`
`supports-auto-init`	Boolean	Indicates whether a DPE supports automatic initialization of a default context. **Affected By** - `supports-default-context` - `supports-initialize-context`
`supports-simulation`	Boolean	Indicates whether a DPE supports simulation contexts. **Affected By** - `supports-context-handles`
`supports-cdi-export`	Boolean	Indicates whether a DPE supports CDI export as part of the `DeriveContext` operation. **Implications** - If false, `cdi-export-format` is irrelevant and can be omitted
`supports-recursive-derivation`	Boolean	Indicates whether a DPE supports recursive derivation. **Implications** - If false, `recursive-derivation` is irrelevant and can be omitted

Use Cases		
supports-signing	Boolean	Indicates whether a DPE supports signing use cases. **Implications** • If true, supports-sign MUST be true • If false, supports-certify-key, supports-sign, and supports-certificates MUST be false • If false and supports-asymmetric-unseal is also false, asymmetric-derivation is irrelevant and can be omitted • If false and supports-symmetric-sign is also false, symmetric-derivation is irrelevant and can be omitted • At least one of supports-signing or supports-sealing MUST be true **Affected By** • supports-sealing • supports-certify-key • supports-sign
supports-sealing	Boolean	Indicates whether a DPE supports sealing use cases. **Implications** • If true, supports-unseal MUST be true • If false, the following MUST also be false: ○ supports-seal ○ supports-unseal ○ supports-sealing-public ○ supports-unseal-policy ○ supports-asymmetric-unseal • At least one of supports-sealing or supports-signing MUST be true **Affected By** • supports-signing • supports-seal • supports-unseal • supports-sealing-public
Commands[1]		
supports-get-profile	Boolean	Indicates whether a DPE supports the GetProfile command.
supports-open-session	Boolean	Indicates whether a DPE supports the OpenSession command. **Implications**

[1] Note: mandatory commands not represented in this section.

		• If true, `supports-encrypted-sessions` MUST be true **Affected By** • `supports-encrypted-sessions`
`supports-close-session`	Boolean	Indicates whether a DPE supports the `CloseSession` command. **Implications** • If true, `supports-encrypted-sessions` MUST be true **Affected By** • `supports-encrypted-sessions`
`supports-sync-session`	Boolean	Indicates whether a DPE supports the `SyncSession` command. **Implications** • If true, `supports-encrypted-sessions` MUST be true **Affected By** • `supports-encrypted-sessions`
`supports-initialize-context`	Boolean	Indicates whether a DPE supports the `InitializeContext` command. **Implications** • If false, `supports-default-context` and `supports-auto-init` MUST be true
`supports-get-certificate-chain`	Boolean	Indicates whether a DPE supports the `GetCertificateChain` command. **Implications** • If true, `supports-certificates` and `supports-signing` MUST be true **Affected By** • `supports-signing` • `supports-certificates`
`supports-certify-key`	Boolean	Indicates whether a DPE supports the `CertifyKey` command. **Implications** • If true, `supports-certificates` and `supports-signing` MUST be true **Affected By** • `supports-signing` • `supports-certificates`

supports-sign	Boolean	Indicates whether a DPE supports the Sign command. **Implications** • If true, supports-signing MUST be true • If false, supports-signing MUST be false • If false, the following are irrelevant and can be omitted: ○ to-be-signed-format ○ signature-format ○ supports-symmetric-sign **Affected By** • supports-signing
supports-seal	Boolean	Indicates whether a DPE supports the Seal command. **Implications** • If true, supports-sealing MUST be true **Affected By** • supports-sealing
supports-unseal	Boolean	Indicates whether a DPE supports the Unseal command. **Implications** • If true, supports-sealing MUST be true • If false, supports-asymmetric-unseal and supports-sealing MUST be false **Affected By** • supports-sealing
supports-sealing-public	Boolean	Indicates whether a DPE supports the DeriveSealingPublicKey command. **Implications** • If true, supports-sealing MUST be true • If false, supports-asymmetric-unseal MUST be false **Affected By** • supports-sealing • supports-asymmetric-unseal
supports-rotate-context-handle	Boolean	Indicates whether a DPE supports the RotateContextHandle command.
Derivation		
dice-derivation	String	Names a scheme for how input data is mixed with a UDS or CDI to produce a new CDI. The scheme MUST define:

		• Cryptographic algorithms • A deterministic process from input to CDI
`asymmetric-derivation`	String	Names a scheme for how an asymmetric key pair is derived from a CDI. The scheme MUST define: • Cryptographic algorithms including: ○ Key type, size, and domain parameters, for both signing and sealing ○ Signature scheme ○ Asymmetric sealing scheme • A deterministic derivation process from CDI to key pair for all supported derivations. For example, ECA keys, signing keys, and sealing keys. **Affected By** • `supports-signing` • `supports-asymmetric-unseal`
`symmetric-derivation`	String	Names a scheme for how a symmetric key is derived from a CDI. The scheme MUST define: • Cryptographic algorithms including: ○ Key type and size for both signing and sealing ○ Signature scheme ○ Sealing scheme • A deterministic derivation process from CDI to key material for all supported derivations. For example, MAC keys and sealing keys. **Affected By** • `supports-sealing` • `supports-symmetric-sign`
`supports-any-label`	Boolean	Indicates whether a DPE allows arbitrary labels or a fixed set of labels. If a DPE supports only a fixed set, the supported-labels attribute defines that fixed set. **Implications** • If true, `supported-labels` is irrelevant and can be omitted • If false, `supported-labels` MUST be provided
`supported-labels`	String	Names a fixed set of labels that are supported by the DPE. The list MUST specify for each label the exact bytes supported as a label argument. For example, if the list is described using text strings, the encoding must be specified as well. **Affected By** • `supports-any-label`
`initial-derivation`	String	Names a scheme for deriving an initial state, for example a UDS or CDI(s), when a context is initialized. This can incorporate the

		seed argument to the InitializeContext command and/or values available internally to the DPE. The scheme MUST cover: • Cryptographic algorithms • A deterministic process from seed(s) to UDS or CDI(s) and certificates if applicable • Whether the seed argument is used and, if so: o The format of the seed argument o Any requirements on the seed argument for secure operation
`recursive-derivation`	String	Names a scheme for how contexts are affected as part of recursive derivation. The scheme MUST cover: • Any variation from the `dice-derivation` scheme, if any • Any variation from the `eca-certificate-format`, if any **Affected By** • `supports-recursive-derivation`
Input		
`input-format`	String	Names a scheme for how input data is formatted when passed to the `DeriveContext` command. The scheme MUST define: • Unambiguous definition of the structure of the data: o As it appears in the DPE input argument o As it is processed by the DICE derivation • Canonical encoding(s) of the data: o As it appears in the DPE input argument o As it is processed by the DICE derivation • For each field, whether the field is used for the signing derivations and/or sealing derivations, where applicable • Basic attributes of each field, if applicable, including whether it is optional, repeatable, typed, or has value constraints (e.g. max size) How the input is presented in a certificate is defined by the certificate format attributes.
`supports-internal-inputs`	Boolean	Indicates whether a DPE supports internal inputs. **Implications** • If false, `supports-internal-dpe-info` and `supports-internal-dpe-dice` MUST be false • If false, `internal-inputs` is irrelevant and can be omitted
`supports-internal-dpe-info`	Boolean	Indicates whether a DPE supports the `dpe-info` internal input. **Implications** • If false, `internal-dpe-info-type` is irrelevant and can be omitted

		Affected By • `supports-internal-inputs`
`supports-internal-dpe-dice`	Boolean	Indicates whether a DPE supports the `dpe-dice` internal input **Implications** • If false, `internal-dpe-dice-type` is irrelevant and can be omitted **Affected By** • `supports-internal-inputs`
`internal-dpe-info-type`	String	Names a type that defines properties of the `dpe-info` internal input. The definition MUST define the same elements as other internal input definitions. See the `internal-inputs` attribute. **Affected By** • `supports-internal-dpe-info`
`internal-dpe-dice-type`	String	Names a type that defines properties of the `dpe-dice` internal input. The definition MUST define the same elements as other internal input definitions. See the `internal-inputs` attribute. **Affected By** • `supports-internal-dpe-dice`
`internal-inputs`	String	A comma-separated list of internal inputs that are supported. The pre-defined `dpe-info` and `dpe-dice` inputs are not in this list. Each item in the list MUST name an internal input that is well defined. An internal input definition MUST define: • The semantics of the input, or the semantics of each field if the input is composed of multiple fields • The structure of the input • The encoding of the input as it is processed by the DICE derivation • Basic attributes of each field, if applicable, including whether it is optional, repeatable, typed, or has value constraints (e.g. max size) • Whether the input appears in certificates, per field if applicable As with all other inputs, how an internal input is presented in a certificate is defined by the certificate format attributes. **Affected By** • `supports-internal-inputs`
Certificates		

`supports-certificates`	Boolean	Indicates whether a DPE supports generating certificates. **Implications** ● If false, the following MUST also be false: ○ `supports-get-certificate-chain` ○ `supports-certify-key` ○ `supports-certificate-policies` ○ `supports-eca-certificates` ○ `supports-external-key` ● If true, `supports-certify-key` MUST be true ● If false, the following attributes are irrelevant and can be omitted: ○ `max-certificate-size` ○ `max-certificate-chain-size` ○ `appends-more-certificates` ○ `leaf-certificate-format` ● If false and `supports-asymmetric-unseal` is also false, `public-key-format` is irrelevant and can be omitted **Affected By** ● `supports-signing` ● `supports-certify-key`
`max-certificate-size`	Number	The maximum certificate size, in bytes. **Affected By** ● `supports-certificates`
`max-certificate-chain-size`	Number	The maximum number of certificates in a chain, per context. **Affected By** ● `supports-certificates`
`appends-more-certificates`	Boolean	Indicates whether a DPE appends more certificates to each chain after those generated by the DPE. For example, whether a static manufacturer certificate chain is appended that anchors the chain to a self-signed root. **Affected By** ● `supports-certificates`
`supports-certificate-policies`	Boolean	Indicates whether a DPE supports any certificate policies via the `policies` argument to the `CertifyKey` command. **Implications** ● If false, the following MUST also be false: ○ `supports-policy-identity-init` ○ `supports-policy-identity-loc` ○ `supports-policy-attest-init`

		○ `supports-policy-attest-loc` ○ `supports-policy-assert-init` ○ `supports-policy-assert-loc` ● If false, `certificate-policies` is irrelevant and can be omitted **Affected By** ● `supports-certificates`
`supports-policy-identity-init`	Boolean	Indicates whether a DPE supports the `tcg-dice-kp-identityInit` policy. **Affected By** ● `supports-certificate-policies`
`supports-policy-identity-loc`	Boolean	Indicates whether a DPE supports the `tcg-dice-kp-identityLoc` policy. **Affected By** ● `supports-certificate-policies`
`supports-policy-attest-init`	Boolean	Indicates whether a DPE supports the `tcg-dice-kp-attestInit` policy. **Affected By** ● `supports-certificate-policies`
`supports-policy-attest-loc`	Boolean	Indicates whether a DPE supports the `tcg-dice-kp-attestLoc` policy. **Affected By** ● `supports-certificate-policies`
`supports-policy-assert-init`	Boolean	Indicates whether a DPE supports the `tcg-dice-kp-assertInit` policy. **Affected By** ● `supports-certificate-policies`
`supports-policy-assert-loc`	Boolean	Indicates whether a DPE supports the `tcg-dice-kp-assertLoc` policy. **Affected By** ● `supports-certificate-policies`
`certificate-policies`	String	A comma-separated list of policies that are supported. The pre-defined `tcg-*` policies are not in this list. Each item in the list MUST name a policy that is well defined. A policy definition MUST define: ● The semantics of the policy: what it authorizes

		• The identifier of the policy (e.g. ASN.1 OID) • Any changes to how a certificate is generated when this policy is specified • Any implications to other policies or options **Affected By** • `supports-certificate-policies`
`supports-eca-certificates`	Boolean	Indicates whether the `DeriveContext` command supports the `create-certificate` argument set to true, causing an ECA certificate to be created. **Implications** • If false, `eca-certificate-format` is irrelevant and can be omitted **Affected By** • `supports-certificates`
`eca-certificate-format`	String	Names a certificate format that describes how an intermediate certificate is generated in association with a CDI that represents a particular DICE component. This is the certificate that is generated by `DeriveContext` when the `create-certificate` argument is true. The certificate format MUST define: • Certificate structure (e.g. X.509, CWT) • Certificate encoding (e.g. ASN.1 DER, CBOR) • Certificate content (e.g. subject, issuer, policies) • How input data maps to certificate fields • How accumulated certificate information from multiple components is combined and mapped to certificate fields • How, if a CDI was exported from a DPE, this is indicated in a certificate. • How multiple certificates are ordered when represented as a chain, as returned by `GetCertificateChain` **Affected By** • `supports-eca-certificates`
`leaf-certificate-format`	String	Names a certificate format that describes how a leaf certificate is generated in association with a CDI that represents a particular DICE component. This is the certificate that is generated by the `CertifyKey` command. The certificate format MUST define: • Certificate structure (e.g. X.509, CWT) • Certificate encoding (e.g. ASN.1 DER, CBOR) • Certificate content (e.g. subject, issuer, policies) • How accumulated certificate information from multiple components is combined and mapped to certificate fields • For each supported policy for the `CertifyKey policies` argument, how the certificate changes when the policy is

		specified or omitted, and whether the policy has implications on other policies - Behavior when no policies are specified, e.g. the policies argument is omitted when invoking `CertifyKey` - Whether and how the label argument is used in the certificate **Affected By** - `supports-certificates`
Signatures		
`public-key-format`	String	Names a format that describes how public keys are formatted in DPE arguments. The format MUST define: - Public key structure and encoding - Algorithm identification **Affected By** - `supports-certificates` - `supports-asymmetric-unseal`
`supports-external-key`	Boolean	Indicates whether a DPE supports certifying external public keys. If supported, a DPE MUST support certification of any type of public key that can be represented by the public key format. The key type used by a DPE when deriving asymmetric keys internally and the corresponding signature and encryption algorithms are described by the `asymmetric-derivation` attribute. This does not change based on the external key type, so the issuer and subject of the certificate may have different key types. **Affected By** - `supports-certificates`
`to-be-signed-format`	String	Names a format for the `Sign` command `to-be-signed` argument. The format MUST define: - The structure and encoding of the data - Any processing of the data by the DPE **Affected By** - `supports-sign`
`signature-format`	String	Names a format for the signature returned by the `Sign` command. The format MUST define the structure and encoding of the data for all types of supported signatures (i.e., symmetric and asymmetric). **Affected By** - `supports-sign`

supports-symmetric-sign	Boolean	Whether a DPE supports generating symmetric signatures, e.g., MACs. **Affected By** • supports-sign
Sealing		
supports-asymmetric-unseal	Boolean	Indicates whether a DPE supports asymmetric unsealing via the Unseal command and the derivation of public keys for asymmetric sealing. **Implications** • If false, supports-sealing-public MUST be false • If true, supports-sealing-public MUST be true • If false and supports-signing is also false, asymmetric-derivation is irrelevant and can be omitted • If false and supports-certificates is also false, public-key-format is irrelevant and can be omitted **Affected By** • supports-sealing • supports-unseal • supports-sealing-public
supports-unseal-policy	Boolean	Indicates whether a DPE supports an unseal policy. **Implications** • If false, unseal-policy-format is irrelevant **Affected By** • supports-sealing
unseal-policy-format	String	Names a format for the unseal-policy argument for the Seal or DeriveSealingPublicKey command. The format MUST cover: • The structure and canonical encoding of the data • The semantics of the data in terms of how a DPE enforces the policy **Affected By** • supports-unseal-policy
Localities		
supports-multiple-localities	Boolean	Indicates whether a DPE supports multiple localities. **Implications** • If false, locality-id-format is irrelevant and can be omitted

`locality-id-format`	String	Names a format for specifying a target locality. The format MUST cover: - The structure and encoding of the identifier, including a fixed length if applicable - The semantics of the id in terms of how it maps to the underlying platform notion of locality **Affected By** - `supports-multiple-localities`
Export		
`export-cdi-format`	String	Names a format for the exported-cdi output argument of the DeriveContext command. The format MUST cover: - The structure and encoding of the value **Affected By** - `supports-export-cdi`

Table 2: DPE profile attributes

7.3 Sample Profile

Table 3 is an example of a profile. For convenience the definition of named values are provided inline in the table. The cryptographic algorithms chosen target at least 128 bits of strength. Other profiles can use the names defined in this section, provided the definition remains as specified here.

Attribute	Description
General	
`name`	tcg.sample.1
`inherits`	Empty
`dpe-spec-version`	1
`max-message-size`	65535
`uses-multi-part-messages`	False
`supports-concurrent-operations`	NA
Sessions	
`supports-encrypted-sessions`	True
`supports-derived-sessions`	True

`max-sessions`	Unlimited
`session-protocol`	tcg.protocol.noise-nk The protocol tcg.protocol.noise-nk is defined as follows: ● Noise_NK_25519_AESGCM_SHA256 for initial handshake and transport ● Noise_NNpsk0_25519_AESGCM_SHA256 for derived handshake ● The PSK for the derived handshake is the channel binding token of the existing session ● In both handshakes, the session ID is contained in the responder payload field ● Each session can send up to 2^64 ciphertext messages
`supports-session-sync`	True
`session-sync-policy`	tcg.monotonic-sync The policy tcg.monotonic-sync is defined as requiring only that the new value of the counter is greater than or equal to the existing value of the counter.
Contexts	
`supports-default-context`	True
`supports-context-handles`	True
`max-contexts-per-session`	Unlimited
`max-context-handle-size`	Unlimited
`supports-auto-init`	False
`supports-simulation`	True
`supports-cdi-export`	True
`supports-recursive-derivation`	True
Use Cases	
`supports-signing`	True
`supports-sealing`	True
Commands[2]	
`supports-get-profile`	True

[2] Note: mandatory commands not represented in this section.

`supports-open-session`	True
`supports-close-session`	True
`supports-sync-session`	True
`supports-initialize-context`	True
`supports-get-certificate-chain`	True
`supports-certify-key`	True
`supports-sign`	True
`supports-seal`	True
`supports-unseal`	True
`supports-sealing-public`	True
`supports-rotate-context-handle`	True
Derivation	
`dice-derivation`	tcg.derive.hkdf-sha256 The scheme tcg.derive.hkdf-sha256 is defined as follows: ● The algorithm is HKDF with SHA256 as the Hash option ● The scheme produces a CDI using HKDF with these arguments: ○ length (L): 32 ○ input key material (IKM): the DICE secret (UDS or CDI) ○ information: the input, encoded for derivation ○ salt: ASCII encoded string of “CDI_Sign” for a signing CDI or “CDI_Seal” for a sealing CDI
`asymmetric-derivation`	tcg.derive.hkdf-sha256-curve25519 The scheme tcg.derive.hkdf-sha256-curve25519 is defined as follows: ● The derivation algorithm is HKDF with SHA256 as the Hash option ● The asymmetric key type is curve25519 ● Signature scheme is Ed25519 ● Sealing scheme is hybrid encryption using DHKEM(X25519, HKDF-SHA256) + HKDF-SHA256 + AES-256-GCM ● The HKDF information argument is a SHA256 hash of the label argument, if applicable, even if the label is empty. When there is no notion of label, leave the information empty (zero bytes). ● The derivation scheme produces a 32-byte private key using HKDF with these arguments: ○ length (L): 32

	○ input key material (IKM): the CDI ○ information: the label-based information value ○ salt: SHA256 hash of one of the following ASCII encoded strings, without a null terminator: ■ "Key_Pair_25519_Sign" for signing ■ "Key_Pair_25519_Seal" for sealing ■ "Key_Pair_25519_ECA" for an embedded CA key
`symmetric-derivation`	tcg.derive.hkdf-sha256-aes256-gcm-siv-hmac-sha256 The scheme tcg.derive.hkdf-sha256-aes256-gcm-hmac-sha256 is defined as follows: ● The derivation algorithm is HKDF with SHA256 as the Hash option ● The HKDF information argument is a SHA256 hash of the label argument, if applicable, even if the label is empty. When there is no notion of label, information SHALL be empty (zero bytes). ● For sealing, the encryption scheme is AEAD_AES_256_GCM_SIV as defined in RFC 8452. The 96-bit nonce is randomly generated and prepended to the ciphertext. ● For signing, the scheme is HMAC-SHA256 with a 256-bit key ● The sealing and signing keys are derived using these HKDF arguments: ○ length (L): 32 ○ input key material (IKM): the CDI ○ information: the label-based information value ○ salt: SHA256 hash of one of the following ASCII encoded strings, without a null terminator: ▪ "Key_HMAC_Sign" for the signing ▪ "Key_AES_Seal" for the sealing
`supports-any-label`	True
`supported-labels`	Ignored (`supports-any-label`, true)
`initial-derivation`	tcg.init.combined-uds.hkdf-sha256 The scheme tcg.init.combined-uds.hkdf-sha256 is defined as follows: ● The DPE combines an internal seed with the seed argument from the client to derive a UDS that comprises the initial context state ● HKDF-SHA256 is used with these arguments: ○ length (L): 32 ○ input key material (IKM): the internal seed value ○ information: the seed argument value ○ salt: none ● The internal seed meets all requirements for a UDS in terms of entropy and availability ● The seed argument: ○ is a raw value: it is not interpreted ○ is expected to be no more than 32 bytes in length

	○ has no security requirements: it can be empty
`recursive-derivation`	tcg.derive.recursive.default The scheme tcg.derive.recursive.default is defined as follows: ● No changes to the CDI derivation or certificate format ● Original input is retained for each context and CDIs are recursively recomputed from scratch from the new parent CDI and the original input ● Only the context given to the DeriveContext command directly incorporates the given input
Input	
`input-format`	tcg.format.tcb-info The scheme "tcg.format.tcb-info" is defined as follows: ● The input is a DiceTcbInfo ASN.1 structure as defined by [3] ● The input is encoded using ASN.1 DER for DPE input, derivation, and certificates ● All fields are used for a signing derivation ● All fields except FWIDs are used for a sealing derivation
`supports-internal-inputs`	True
`supports-internal-dpe-info`	True
`supports-internal-dpe-dice`	True
`internal-dpe-info-type`	tcg.basic-dpe-info The tcg.basic-dpe-info type is defined as follows: ● The information is a profile descriptor, exactly as it would be returned by the `GetProfile` command in terms of semantics, structure, and encoding ● The information appears in its entirety in certificates
`internal-dpe-dice-type`	tcg.basic-dpe-dice The tcg.basic-dpe-dice type is defined as follows: ● There are two elements: (1) a CDI and (2) a certificate chain that represent the current identity of the DPE ● The CDI is an input for a signing derivation, but does not appear in the certificate ● The certificate chain is not included in a derivation, but appears in the certificate, if applicable ● The semantics, structure, and encoding of the certificate chain are entirely implementation-dependent and help from the DPE vendor is necessary to parse/verify it

`internal-inputs`	Empty
Certificates	
`supports-certificates`	True
`max-certificate-size`	Unlimited
`max-certificate-chain-size`	Unlimited
`appends-more-certificates`	False
`supports-certificate-policies`	True
`supports-policy-identity-init`	True
`supports-policy-identity-loc`	True
`supports-policy-attest-init`	True
`supports-policy-attest-loc`	True
`supports-policy-assert-init`	False
`supports-policy-assert-loc`	False
`certificate-policies`	Empty
`supports-eca-certificates`	True
`eca-certificate-format`	tcg.certificate.basic-eca The profile tcg.certificate.basic-eca is designed to work with the tcg.format.tcb-info input format and is defined as follows: ● An X.509 ECA certificate as defined by [4] ● The key usage field MUST contain only keyCertSign ● The basic constraints path length should be set to zero if the `allow-new-context-to-derive` argument to `DeriveContext` is false, otherwise omitted ● The `tcg-dice-kp-eca` policy MUST be the only `tcg-*` policy ● One `tcg-dice-TcbInfo` extension MUST be added using the TcbInfo from the `input-data` argument, and one additional `tcg-dice-TcbInfo` extension MUST be added for each accumulated TcbInfo from previous invocations of `DeriveContext` with `create-certificate` set to false, if any ● If internal inputs were indicated in the `internal-inputs` argument to `DeriveContext`, a `tcg-dice-DpeInternal` extension (OID value 2.23.133.5.4.200.1) MUST be added whose content is a CBOR map with each key being an `internal-input-type` value (e.g., `dpe-info` = 1) and each value being the corresponding data as bytes

	● If internal inputs have been accumulated, one additional `tcg-dice-DpeInternal` extension MUST be added for each set, i.e., for each time `DeriveContext` was called with internal inputs selected and `create-certificate` set to false. ● When in a chain, certificates are ordered leaf to root.
`leaf-certificate-format`	tcg.certificate.basic-leaf The profile tcg.certificate.basic-leaf is designed to work with the tcg.format.tcb-info input format and is defined as follows: ● An X.509 certificate meeting the requirements of [4] according to the `CertifyKey` policies argument ● If multiple policies are specified, the certificate contains the policy OID for each and MUST meet the requirements for each corresponding certificate type, e.g. if the `tcg-dice-kp-attestLoc` policy is set, an attestation certificate is generated ● If no policies are selected, the `tcg-dice-kp-attestLoc` policy is used as a default ● The key usage field MUST contain only digitalSignature ● The certificate MUST NOT contain basic constraints ● The `tcg-dice-kp-eca` policy MUST NOT be included ● One `tcg-dice-TcbInfo` extension MUST be added for each accumulated TcbInfo from previous invocations of `DeriveContext` with `create-certificate` set to false, if any ● One `tcg-dice-DpeInternal` extension MUST be added for each accumulated set of internal inputs ● The label is not used in the certificate
Signatures	
`public-key-format`	tcg.key-format.x509 The format tcg.key-format.x509 uses the X.509 SubjectPublicKeyInfo ASN.1 sequence as defined by [11]. It is encoded using ASN.1 DER.
`supports-external-key`	True
`to-be-signed-format`	tcg.tbs-format.raw The format tcg.tbs-format.raw is defined as opaque raw bytes that will be signed directly using the signing scheme with no additional processing.
`signature-format`	tcg.signature.raw The format tcg.signature.raw uses raw signature bytes as defined by the signature scheme.
`supports-symmetric-sign`	True

Sealing	
`supports-asymmetric-unseal`	True
`supports-unseal-policy`	True
`unseal-policy-format`	tcg.unseal-policy.tcb-info-layer-svn-clamp This policy is used to limit which component versions are allowed to unseal. The policy format is a CBOR map where the keys each identify a component and the value is an integer value indicating the minimum version of that component. At the time of unseal, if and only if, for each component that appears in the map, the current version of the component is greater than or equal to the minimum version that appears in the policy, the unseal will be allowed. The values of the component identifiers correspond to the 'layer' field of the DiceTcbInfo structure and the value of the versions correspond to the 'svn' field of the DiceTcbInfo structure. Note, this type of policy is limited to systems where a single component per layer participates in the policy.
Localities	
`supports-multiple-localities`	False
`locality-id-format`	Ignored (`supports-multiple-localities`, false)
Export	
`cdi-export-format`	tcg.cdi-export.raw The tcg.cdi-export.raw format exports the CDI as a single, raw, fixed size value.

Table 3: Sample DPE profile attributes

7.4 Profile Descriptors

A profile descriptor describes all attributes of a profile. Like a profile, a profile descriptor is *complete*. A profile descriptor can describe a profile by name or by full attribute list. A profile descriptor MUST include the full attribute list when a profile has no name. A profile descriptor SHOULD include the full attribute list when a profile name is not well known to all potential clients.

A profile descriptor is encoded as a CBOR map with each attribute assigned a key. If a profile is described by name, the name MUST be the only item in the map. A profile descriptor MAY be tagged using the CBOR tag 1146111423. When returned by a `GetProfile` command a DPE MUST NOT tag the descriptor.

The descriptor format is as follows:

```
profile-descriptor = {
  * $attribute-bool => bool,
  * $attribute-number => uint,
  * $attribute-string => tstr,
  ? &(inherits: 0) => bytes,
  * tstr => any
}
$attribute-string /= &(name: 1)
$attribute-number /= &(dpe-spec-version: 2)
$attribute-number /= &(max-message-size: 3)
$attribute-bool /= &(uses-multi-part-messages: 4)
$attribute-bool /= &(supports-concurrent-operations: 5)
$attribute-bool /= &(supports-encrypted-sessions: 6)
$attribute-bool /= &(supports-derived-sessions: 7)
$attribute-number /= &(max-sessions: 8)
$attribute-string /= &(session-protocol: 9)
$attribute-bool /= &(supports-session-sync: 10)
$attribute-string /= &(session-sync-policy: 11)
$attribute-bool /= &(supports-default-context: 14)
$attribute-bool /= &(supports-context-handles: 15)
$attribute-number /= &(max-contexts-per-session: 16)
$attribute-number /= &(max-context-handle-size: 17)
$attribute-bool /= &(supports-auto-init: 18)
$attribute-bool /= &(supports-simulation: 19)
$attribute-bool /= &(supports-signing: 20)
$attribute-bool /= &(supports-sealing: 21)
$attribute-bool /= &(supports-get-profile: 22)
$attribute-bool /= &(supports-open-session: 23)
$attribute-bool /= &(supports-close-session: 24)
$attribute-bool /= &(supports-sync-session: 25)
$attribute-bool /= &(supports-init-context: 28)
```

```
$attribute-bool /= &(supports-certify-key: 29)
$attribute-bool /= &(supports-sign: 30)
$attribute-bool /= &(supports-seal: 31)
$attribute-bool /= &(supports-unseal: 32)
$attribute-bool /= &(supports-sealing-public: 33)
$attribute-bool /= &(supports-rotate-context-handle: 34)
$attribute-string /= &(dice-derivation: 35)
$attribute-string /= &(asymmetric-derivation: 36)
$attribute-string /= &(symmetric-derivation: 37)
$attribute-bool /= &(supports-any-label: 38)
$attribute-string /= &(supported-labels: 39)
$attribute-string /= &(initial-derivation: 40)
$attribute-string /= &(input-format: 41)
$attribute-bool /= &(supports-internal-inputs: 42)
$attribute-bool /= &(supports-internal-dpe-info: 43)
$attribute-bool /= &(supports-internal-dpe-dice: 44)
$attribute-string /= &(internal-dpe-info-type: 45)
$attribute-string /= &(internal-dpe-dice-type: 46)
$attribute-string /= &(internal-inputs: 47)
$attribute-bool /= &(supports-certificates: 48)
$attribute-number /= &(max-certificate-size: 49)
$attribute-number /= &(max-certificate-chain-size: 50)
$attribute-bool /= &(appends-more-certificates: 51)
$attribute-bool /= &(supports-certificate-policies: 52)
$attribute-bool /= &(supports-policy-identity-init: 53)
$attribute-bool /= &(supports-policy-identity-loc: 54)
$attribute-bool /= &(supports-policy-attest-init: 55)
$attribute-bool /= &(supports-policy-attest-loc: 56)
$attribute-bool /= &(supports-policy-assert-init: 57)
$attribute-bool /= &(supports-policy-assert-loc: 58)
```

```
$attribute-string /= &(certificate-policies: 59)
$attribute-bool /= &(supports-eca-certificates: 60)
$attribute-string /= &(eca-certificate-format: 61)
$attribute-string /= &(leaf-certificate-format: 62)
$attribute-string /= &(public-key-format: 63)
$attribute-bool /= &(supports-external-key: 64)
$attribute-string /= &(to-be-signed-format: 65)
$attribute-string /= &(signature-format: 66)
$attribute-bool /= &(supports-symmetric-sign: 67)
$attribute-bool /= &(supports-asymmetric-unseal: 68)
$attribute-bool /= &(supports-unseal-policy: 69)
$attribute-string /= &(unseal-policy-format: 70)
$attribute-bool /= &(supports-multiple-localities: 71)
$attribute-string /= &(locality-id-format: 72)
$attribute-bool /= &(supports-get-certificate-chain: 73)
```