Global Platform®

The standard for secure digital services and devices

GlobalPlatform Technology

Entity Attestation Protocol Specification

Version 0.0.0.26

Public Review #2

July 2024

Document Reference: GPP_SPE_001

THIS SPECIFICATION OR OTHER WORK PRODUCT IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NON-INFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY IMPLEMENTATION OF THIS SPECIFICATION OR OTHER WORK PRODUCT SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE COMPANY, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER DIRECTLY OR INDIRECTLY ARISING FROM THE IMPLEMENTATION OF THIS SPECIFICATION OR OTHER WORK PRODUCT.

Contents

Tables

Figures

1 INTRODUCTION

The aim of this document is to define Claims and how to assemble, encrypt, and sign them for use in Attestation.

1.1 Audience

This document is intended primarily for the use of device manufacturers aiming to implement attestation procedures, and for developers of applications that rely on the information provided by such devices.

1.2 IPR Disclaimer

Attention is drawn to the possibility that some of the elements of this GlobalPlatform specification or other work product may be the subject of intellectual property rights (IPR) held by GlobalPlatform members or others. For additional information regarding any such IPR that have been brought to the attention of GlobalPlatform, please visit https://globalplatform.org/specifications/ip-disclaimers/. GlobalPlatform shall not be held responsible for identifying any or all such IPR, and takes no position concerning the possible existence or the evidence, validity, or scope of any such IPR.

1.3 References

The tables below list references applicable to this specification. The latest version of each reference applies unless a publication date or version is explicitly stated.

Table 1-1: Normative References

Standard / Specification	Description	Ref
GPC_SPE_034	GlobalPlatform Technology Card Specification v2.3.1	[GPCS]
GP_REQ_025	GlobalPlatform Technology Root of Trust Definitions and Requirements v1.1	[RoT]
GPC_SPE_095	GlobalPlatform Technology Digital Letter of Approval v1.0	[DLoA]
GPD_SPE_010	GlobalPlatform Technology TEE Internal Core API Specification v1.3	[TEE Core]
GPD_SPE_120	GlobalPlatform Technology TEE Management Framework including ASN.1 Profile v1.1.2	[TMF]
GPP_SPE_009	GlobalPlatform Technology TPS Client API Specification v1.0	[TPS Client]
IETF RFC 2119	Key words for use in RFCs to Indicate Requirement Levels	[RFC 2119]
IETF RFC NNNN	Entity Attestation Token Pending publication (draft-ietf-rats-eat-29)	[draft-ietf-rats-eat]

Standard / Specification	Description	Ref
IETF RFC NNNN	Unprotected CWT Claims Sets Pending publication (draft-ietf-rats-uccs-10)	[draft-birkholz-rats-uccs]
IETF RFC 4122	A Universally Unique Identifier (UUID)	[RFC 4122]
IETF RFC 6979	Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)	[RFC 6979]
IETF RFC 8949	Concise Binary Object Representation (CBOR)	[RFC 8949]
IETF RFC 8152	CBOR Object Signing and Encryption (COSE)	[RFC 8152]
IETF RFC 8230	Using RSA Algorithms with COSE Messages	[RFC 8230]
IETF RFC 8392	CBOR Web Token (CWT)	[RFC 8392]
IETF RFC 8520	Manufacturer Usage Description (MUD)	[RFC 8520]
IETF RFC 8610	Concise Data Definition Language (CDDL)	[RFC 8610]
ISO/IEC 7812-1:2017	Identification cards -- Identification of issuers -- Part 1: Numbering system	[ISO 7812-1]

1.4 Terminology and Definitions

The following meanings apply to SHALL, SHALL NOT, MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY in this document (refer to [RFC 2119]):

- **SHALL** indicates an absolute requirement, as does **MUST**.
- **SHALL NOT** indicates an absolute prohibition, as does **MUST NOT**.
- **SHOULD** and **SHOULD NOT** indicate recommendations.
- **MAY** indicates an option.

Selected terms used in this document are included in Table 1-2.

Table 1-2: Terminology and Definitions

Term	Definition
Application	Device/terminal/mobile application. An application that is installed in and runs within the device.
Application Programming Interface (API)	A set of rules that software programs can follow to communicate with each other.
Attestation	The process of providing information about credentials within an Entity, with some level of assurance of authenticity.
Basic Parcel Exchange	The generic Operation (described in section 5.2) on which more specific Operations are based.
Bootstrapped Root of Trust	A Root of Trust whose implementation required several components. It is composed of one Initial Root of Trust Component and one or more Extended Root of Trust Components.

Term	Definition
Chain of Trust	A transitive trust relationship starting from a Root of Trust that is propagated to the Validated/Measured Modules (as discussed in [RoT]) when a software module verifies/measures the next software module and keeps a reportable record of this verification.
Claim	A piece of information in the form of a key/value pair that represents an assertion about a single characteristic either internal to an Entity or provided to it by a third party.
Claim Key	The CBOR map key used to identify a Claim.
Claim Name	The human-readable name used to identify a Claim.
Claim Value	The CBOR map value representing the value of a Claim.
Claimant	A producer of evidence about its own characteristics.
Claims Set	An aggregation of one or more Claims.
Detached EAT Bundle	Optional structure that can be used to convey claim sets for which a digest is included in an EAT.
Device	An end-user product that includes at least one Platform.
Empty EAP	In the context of this document, an Entity Attestation Parcel with an empty Claims Set.
Endorse	To provide additional information – signature, encryption, or MAC – to enhance the integrity, confidentiality, or authenticity of the underlying information.
Entity	A Platform that has the capability to generate Claims Entity Attestation Parcels.
Entity Attestation Client (EAC)	A service within an Entity that is responsible for generating a Request Entity Attestation Parcel.
Entity Attestation Parcel (EAP)	An aggregation of credential information that is provided to a Relying Party; can be either endorsed or unendorsed.
Entity Attestation Service (EAS)	A service within an Entity that is responsible for generating a Response Entity Attestation Parcel.
Entity Attestation Token (EAT)	An endorsed aggregation of credential information that is provided to a Relying Party.
Evidence	A piece of information that is included in a Claim.
Execution Environment (EE)	An environment that hosts and executes software. This could be a REE, with hardware hosting Android, Linux, Windows, an RTOS, or other software; it could be a Secure Element or a TEE.
Key Construction Algorithm	A cryptographic mechanism by which a key can be constructed from pre-existing or supplied key material. Key construction can include key agreement and/or key derivation.
Key Distribution	A mechanism by which cryptographic key material can be provided from one device to another by secure means. This can be achieved by Key Construction or Key Wrapping.

Term	Definition
Key Wrapping	A mechanism by which a private key can be provided to a remote device by encrypting it with key material known by both parties.
Message Authentication Code (MAC)	A symmetric cryptographic transformation of data that provides data origin authentication and data integrity. In the context of this specification, created as described in section 12.3.2.
Non-Bootstrapped Root of Trust	A Root of Trust that is composed of only one block of code.
Operation	A defined mechanism by which an Entity Attestation Client can obtain attestation information from an Entity Attestation Service; takes the form of challenge and response. Formally, because an EAT is constructed, transported, and processed in each direction, an Operation is comprised of a Remote Attestation from Client to Service, followed by a Remote Attestation from Service to Client.
Platform	One computing engine and executable code that provides a set of functionalities. SE, TEE, and REE are examples of platforms.
Regular Execution Environment (REE)	An Execution Environment comprising at least one Regular OS and all other components of the device (IC packages, other discrete components, firmware, and software) that execute, host, and support the Regular OSes (excluding any Secure Components included in the device). From the viewpoint of a Secure Component, everything in the REE is considered untrusted, though from the Regular OS point of view there may be internal trust structures. (Formerly referred to as a *Rich Execution Environment (REE)*.) Contrast *Trusted Execution Environment (TEE).*
Regular OS	An OS executing in a Regular Execution Environment. May be anything from a large OS such as Linux down to a minimal set of statically linked libraries providing services such as a TCP/IP stack. (Formerly referred to as a *Rich OS* or *Device OS*.)
Relying Party	In the context of this document, a party that receives an Entity Attestation Token, potentially in response to a request, in order to determine the value or authenticity of credentials contained within it.
Remote Attestation	The process by which attestation information is assembled by an Entity Attestation Service, transferred to a Relying Party, and processed.
Request EAP	An EAP sent from the Entity Attestation Client to the Entity Attestation Service.
Response EAP	An EAP returned by the Entity Attestation Service to the Entity Attestation Client.
Root of Trust (RoT)	A computing engine, code, and possibly data, all co-located on the same platform; provides security services. No ancestor entity is able to provide a trustable attestation (in digest or other form) for the initial code and data state of the Root of Trust. Depending on the implementation, the Root of Trust is either a Bootstrapped or a Non-Bootstrapped Root of Trust.

Term	Definition
Secure Component	A security hardware/firmware combination that acts as an on-device trust anchor. Facilitates collaboration between service providers and device manufacturers, empowering them to ensure adequate security within all devices to protect against threats. Examples include GlobalPlatform's Secure Element and Trusted Execution Environment.
Secure Element (SE)	A tamper-resistant secure hardware component that is used in a device to provide the security, confidentiality, and multiple application environment required to support various business models. May exist in any form factor, such as embedded or integrated SE, SIM/UICC, smart card, smart microSD, etc. For more information, see [GPCS] and related specifications.
Subclaim	In the context of this document, a Claim that has a defined meaning only when included in the value field of a parent Claim.
submodule (submod)	A grouping of information, such as Claims, describing a subsystem within a device. For more information, see [draft-ietf-rats-eat] and section 7.2 of this specification.
TPS Client	An entity that uses the TPS Client API to discover and communicate with a TPS Service. A TPS Client can be either an Application or another TPS Service.
TPS Service	A service in a Secure Component, providing a service to entities in the operating system; accessed using a TPS Service Protocol that is specified in a TPS Service specification.
TPS Service Protocol	A protocol that is used to communicate with the TPS Service; consists of a set of TPS Operations.
Trusted Execution Environment (TEE)	An Execution Environment that runs alongside but isolated from Execution Environments outside of the TEE. A TEE has security capabilities and meets certain security-related requirements: It protects TEE assets against a set of defined threats which include general software attacks as well as some hardware attacks, and defines rigid safeguards as to data and functions that a program can access. There are multiple technologies that can be used to implement a TEE, and the level of security achieved varies accordingly. Contrast *Regular Execution Environment (REE)*.
Trusted OS	An OS executing in a Secure Component. Contrast *Regular OS*.
Unendorsed Claims Set	An unendorsed aggregation of credential information that is provided to a Relying Party.

28

1.5 Abbreviations and Notations

Table 1-3: Abbreviations and Notations

Abbreviation / Notation	Meaning
0 - 9	Decimal digits are not enclosed in quotation marks.
'0' - '9' and 'A' - 'F'	Hexadecimal values are enclosed in straight single quotation marks.
ABI	Application Binary Interface
AES	Advanced Encryption Standard
API	Application Programming Interface
BCD	Binary Coded Decimal
CASD	Controlling Authority Security Domain
CBOR	Concise Binary Object Representation
CCI	Card Capability Information
CCM	Card Content Management
CIN	Card Image Number
COSE	CBOR Object Signing and Encryption
CRD	Card Recognition Data
CWT	CBOR Web Token
DAP	Data Authentication Pattern
DLoA	Digital Letter of Approval
DSA	Digital Signature Algorithm
EAC	Entity Attestation Client
EAP	Entity Attestation Parcel
EAS	Entity Attestation Service
EAT	Entity Attestation Token
ECDSA	Elliptic Curve Digital Signature Algorithm
EE	Execution Environment
eSE	Embedded Secure Element
GCM	Galois Counter Mode
HLOS	High Level Operating System
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IIN	Issuer Identification Number
ISD	Issuer Security Domain
JOSE	JavaScript Object Signing and Encryption

Abbreviation / Notation	Meaning
JSON	JavaScript Object Notation
JWT	JSON Web Token
LFDB	Load File Data Block
MAC	Message Authentication Code
OAEP	Optimal Asymmetric Encryption Padding
PKI	Public Key Infrastructure
PSS	Probabilistic Signature Scheme
RATS	Remote ATtestation ProcedureS
REE	Regular Execution Environment
RoT	Root of Trust
RSA	Rivest / Shamir / Adleman asymmetric algorithm
SE	Secure Element
SHA	Secure Hash Algorithm
SSD	Supplementary Security Domain
submod	submodule
SW	Status Word
TEE	Trusted Execution Environment
UCCS	Unprotected CWT Claims Set
UCS	Unendorsed Claims Set
UEID	Universal Entity ID
URI	Uniform Resource Identifier
UTF-8	Unicode Transformation Format – 8-bit

31

1.6 Revision History

GlobalPlatform technical documents numbered *n*.0 are major releases. Those numbered *n*.1, *n*.2, etc., are minor releases where changes typically introduce supplementary items that do not impact backward compatibility or interoperability of the specifications. Those numbered *n.n*.1, *n.n*.2, etc., are maintenance releases that incorporate errata and clarifications; all non-trivial changes are indicated, often with revision marks.

Table 1-4: Revision History

Date	Version	Description
May 2019	0.0.0.3	Committee Review
Nov 2020	0.0.0.11	Member Review
Feb 2023	0.0.0.19	Member Review #2
July 2023	0.0.0.20	Public Review
July 2024	0.0.0.26	Public Review #2
TBD	1.0	Initial release

2 OVERVIEW

The GlobalPlatform Root of Trust Definitions and Requirements document ([RoT]) describes the concept of a Platform that contains a computing engine and executable code that provides a set of functionalities. Examples of a Platform are a Trusted Execution Environment (TEE) and an embedded Secure Element (eSE). In the context of a Chain of Trust, a Platform contains a Root of Trust.

The IETF NNNN specification ([draft-ietf-rats-eat]) introduces the concept of Attestation, which is a mechanism by which an Entity can provide information about itself with some level of assurance to a Relying Party. The term Entity in this specification corresponds to the term Platform in [RoT]. During Attestation, an Entity Attestation Service builds a set of Claims, each of which consists of a piece of evidence provided by a Claimant. There can be multiple Claimants providing evidence to an Entity Attestation Service, and the EAS itself can also be a Claimant.

There is a need to keep the size of the code in a Root of Trust to a minimum, and to limit the amount of data that has to be transmitted over potentially bandwidth- and power-constrained links. For these reasons, this specification only defines the use of COSE ([RFC 8152]) for the construction of Entity Attestation Tokens. COSE is compact, but has all the necessary features to support both the claims structure and a wide variety of different protection mechanisms including signing and encryption.

The Claims are aggregated, optionally signed using a signing algorithm, optionally encrypted using an encryption algorithm, and optionally MACed using a MAC algorithm; the result is an Entity Attestation Token. This Token is provided to the Entity Attestation Client within the Relying Party, which can then determine the necessary information about the Platform from the Claims and decide its level of confidence in the authenticity of this information. The process of creating an Entity Attestation Token, transferring it, and processing it in an Entity Attestation Client is referred to as Remote Attestation.

There are circumstances in which the communication used to transport claims sets is already sufficiently secure that there is no benefit to further endorsement. In such circumstances, it is possible to avoid the overhead of unnecessary cryptographic algorithms and send a claims set in an unendorsed form. The IETF NNNN specification [draft-birkholz-rats-uccs] introduced the concept of an Unprotected CWT Claims Set, in which the claims set is simply prefixed with a defined value that indicates the nature of the information.

In this document, the term Entity Attestation Parcel (EAP) is used to cover cases in which a claims set is wrapped in either an endorsed (EAT) or unendorsed (UCS) form.

Where a device is composed of multiple platforms, as defined in [RoT], it is possible for a given platform to include Claims from other platforms within the device using the concept of submodules. If the other platform has the ability to create a complete Entity Attestation Token, this can be included as a nested EAT. Otherwise, Claims from the other platform can be nested directly. In the first case, the credibility of the Claims is established by the security level of the platform that generated the nested EAT. In the second case, it is established by the security level of the platform that is including the submodule Claims.

The Entity Attestation Client consists of two conceptually distinct functions: a party that determines which claims it needs in order to decide whether to trust a remote device, and a party that verifies the claims that are received. The verifying party can be local, or it can be delegated to a separate entity. However, since this document defines the mechanisms by which verification can take place, but not the policies that drive where this occurs, it covers both cases.

This specification introduces Operations by which the Entity Attestation Client can request an Entity Attestation Parcel from an Entity Attestation Service using an API. The Relying Party can instruct the Entity Attestation Client to provide various types of information to the Entity Attestation Service, which can influence the content and structure of the Token. An EAP sent from the Entity Attestation Client to the Entity Attestation Service is called a Request EAP, and an EAP returned to the Entity Attestation Client is called a Response EAP.

This specification defines various options that are available for Entity Attestation but makes no restrictions for specific uses. Any such restrictions that assist with interoperability are expected to be defined in configuration documents.

3 PRINCIPLES AND CONCEPTS

This section defines the principles and concepts used or introduced in this specification.

3.1 Entity

An Entity corresponds to a Platform in [RoT] and the terms may be used interchangeably. It contains a Root of Trust, which has the capability to store and/or receive Claims, aggregate them, optionally sign them, optionally encrypt them, and optionally MAC them to produce an Entity Attestation Parcel (EAP, as described in section 3.15).

3.2 Entity Attestation

Entity Attestation is the process of acquiring attestation information by exchanging Entity Attestation Parcels with another Entity. The process of attestation could involve multiple exchanges of Entity Attestation Parcels.

3.2.1 Relying Party

A Relying Party is an application, or a proxy for an application, that wishes to obtain a degree of assurance about the origin, state, and/or attributes of another Entity.

3.2.2 Entity Attestation Client

An Entity Attestation Client is an application within a Relying Party that communicates with an Entity Attestation Service to obtain attestation information on its behalf. The Entity Attestation Client is responsible for evaluating evidence and determining its level of trustworthiness.

3.2.3 Entity Attestation Service

An Entity Attestation Service is a service provided to Relying Parties that receives and parses request Entity Attestation Parcels, and constructs response Entity Attestation Parcels. The Entity Attestation Service assembles claims from its own environment, and/or claims provided by other environments, and optionally uses cryptographic methods to provide authenticity and/or confidentiality.

3.2.4 Remote Attestation

Remote Attestation refers to the complete process in which an Entity Attestation Parcel is created, transferred, and processed by a party distinct (remote) from the party providing evidence.

3.2.5 Entity Attestation Service Protocol

The Entity Attestation Service Protocol is the interface that allows a Relying Party to communicate with an Entity Attestation Service. The Entity Attestation Service Protocol makes use of the Trusted Platform Services Client API ([TPS Client]).

The Entity Attestation Service Protocol is defined in section 4.

3.3 Operation

An exchange of Entity Attestation Parcels using the API is referred to as an Operation.

Operations are defined in section 5.

3.4 Claim

A Claim is an identified piece of evidence either internal to the Root of Trust or received from elsewhere within the Platform. Claims might originate from RoT services such as Measurement or Reporting (as discussed in [RoT]). An example of a Claim originating within the RoT is “here is my boot status providing evidence that I debug is currently disabled”. An example of a Claim not originating within the RoT is “here is a measurement that was provided to me by an installed REE”. For compactness of storage and transfer, a Claim consists of a Claim Key plus a Claim Value. For human readability, a Claim Name is also defined for each Claim.

A Claim can be represented as a pair of CBOR Data Items, as discussed in [RFC 8949].

By defining a Claim for which the value is a compound structure defined below, it is possible to introduce hierarchy (see section 6.1.3.3).

Claims are defined in section 6.

3.5 Claims Set

A Claims Set is an aggregation of one or more Claims. There is no requirement for the Claims to be related in any way, other than by origination. Compound structures such as `submod` SHOULD be used to group claims with the same originator.

A Claims Set SHALL be represented as a CBOR Map, as discussed in [RFC 8949], even when only a single Claim is present.

Rules for Claims Sets are given in section 7.

3.6 Unendorsed Claims Set

An Unendorsed Claims Set is a Claims Set that is prefixed by a defined tag value, but is not encapsulated in COSE format.

Rules for constructing and interpreting Unendorsed Claims Sets are given in section 8.

3.7 COSE Object

At the core of an Entity Attestation Token is a COSE Object as defined in [RFC 8152]. This consists of a protected header, an unprotected header, and a byte string containing the content. In the context of this specification, the content is a Claims Set that is encapsulated in a CBOR Byte String.

3.8 COSE Messages

A COSE Message as defined in [RFC 8152] consists of a COSE Object along with other items specific to the message type. Single and double layer messages are defined for encryption, signing, and the addition of a Message Authentication Code (MAC). COSE Messages can be untagged (`COSE_Untagged_Message`) or tagged (`COSE_Tagged_Message`). A `COSE_Tagged_Message` is a `COSE_Untagged_Message` prepended by the appropriate COSE tag.

3.9 CBOR Web Token

A CBOR Web Token as defined in [RFC 8392] is a COSE Message, optionally prepended by the CWT tag. Given the restrictions imposed by [RFC 8392], this means a CBOR Web Token can take one of three forms.

- A `COSE_Untagged_Message`
- A COSE tag followed by a `COSE_Untagged_Message` (this can also be thought of as a `COSE_Tagged_Message`)
- A CWT tag followed by a COSE tag followed by a `COSE_Untagged_Message` (this can also be thought of as a CWT tag followed by a `COSE_Tagged_Message`)

3.10 Entity Attestation Token

An Entity Attestation Token is a CBOR Web Token (as defined in [RFC 8392] and extended in [draft-ietf-rats-eat] and this specification) that includes an encapsulated Claims Set, and is represented as a COSE Message. A Token can include a signature or a MAC, and can be encrypted.

Rules for Entity Attestation Tokens are given in section 9.

3.11 Signed EAT

A Signed Entity Attestation Token is an EAT which has at least one signature added, and contains suitable information in the protected and unprotected headers (and in the signatures array when present) to enable the receiver to verify the signature.

An untagged signed EAT contains a `COSE_Sign` or a `COSE_Sign1` Message. A tagged signed EAT contains a `COSE_Sign_Tagged` or a `COSE_Sign1_Tagged` message, and is optionally prefixed with the CWT tag.

Rules for constructing and interpreting signed Tokens are given in section 10.

3.12 Encrypted EAT

An Encrypted Entity Attestation Token is an EAT in which the payload has been replaced by ciphertext, and which contains suitable information in the protected and unprotected headers (and in the recipients array when present) to enable the receiver to decrypt the ciphertext. Some algorithms may rely on context information to derive key material. Context information can be known in advance by both parties, transferred as part of the EAT, or a combination of both.

An untagged encrypted EAT contains a `COSE_Encrypt` or a `COSE_Encrypt0` Message. A tagged encrypted EAT contains a `COSE_Encrypt_Tagged` or a `COSE_Encrypt0_Tagged` message, and is optionally prefixed with the CWT tag.

Rules for constructing and interpreting encrypted Tokens are given in section 11.

3.13 MACed EAT

A MACed Entity Attestation Token is an EAT which has an added MAC, and which contains suitable information in the protected and unprotected headers (and in the recipients array when present) to enable the receiver to authenticate the MAC. Some algorithms may rely on context information to derive key material. Context information can be known in advance by both parties, transferred as part of the EAT, or a combination of both.

An untagged MACed EAT contains a `COSE_Mac` or a `COSE_Mac0` Message. A tagged MACed EAT contains a `COSE_Mac_Tagged` or a `COSE_Mac0_Tagged` message, and is optionally prefixed with the CWT tag.

Rules for constructing and interpreting MACed Tokens are given in section 12.

3.14 Tagged EAT

Any Entity Attestation Token can be tagged with values that can help the recipient to identify its nature.

An untagged EAT takes the form of a `COSE_Untagged_Message`. This can be prepended with the appropriate COSE tag value to become a tagged EAT, which takes the form of a `COSE_Tagged_Message`. For example, a tagged unendorsed EAT would contain a `COSE_Sign_Tagged` Message.

A `COSE_Tagged_Message` can also be prepended with the CWT tag value. Note that [RFC 8152] forbids the use of the CWT tag value with a `COSE_Untagged_Message`.

Rules for constructing and interpreting tagged Tokens are given in section 13.

3.15 Entity Attestation Parcel

An Entity Attestation Parcel can take the form of an Entity Attestation Token, an Unendorsed Claims Set, or a Detached EAT Bundle.

3.16 CBOR and CDDL

3.16.1 Normative nature of CDDL definitions

CDDL [RFC 8610] is used to provide normative descriptions of Claims and other structures defined in this document. Where the CDDL definition differs from the text description, the CDDL SHALL prevail.

3.16.2 CDDL prelude

CDDL [RFC 8610] defines a standard prelude. Definitions in this document assume the availability of definitions in the CDDL prelude.

3.16.3 CBOR Deterministic Encoding

Encoders SHOULD generate deterministically encoded CBOR as defined in [RFC 8949].

3.16.4 Validity of CBOR encoding

Encoders MUST NOT generate invalid CBOR data items (as defined in [RFC 8949], section 5.3). Decoders MUST reject an Entity Attestation Parcel containing invalid CBOR item(s).

With specific regard to UTF-8, decoders MUST consider as invalid any string containing invalid UTF-8 byte sequences.

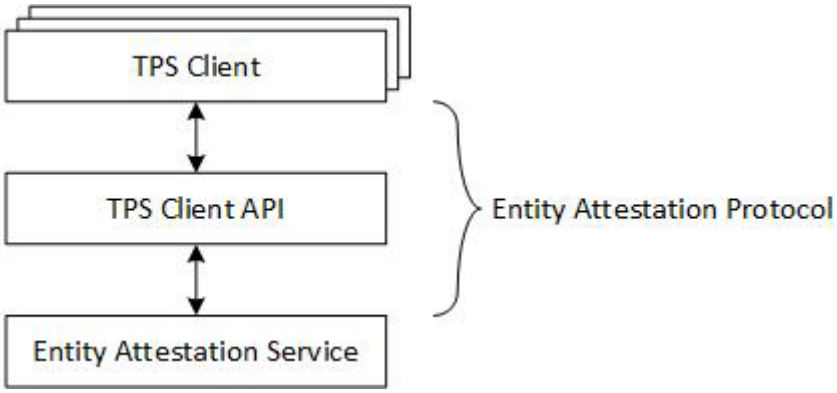
Decoders MUST ignore unknown CBOR items but SHOULD NOT reject an Entity Attestation Parcel containing valid, but unknown CBOR items.

4 ENTITY ATTESTATION API

4.1 Overview

The Entity Attestation API consists of an Entity Attestation Protocol used to convey operational instructions to an Entity Attestation Service implementation via the TPS Client API.

Figure 4-1: Entity Attestation Overview



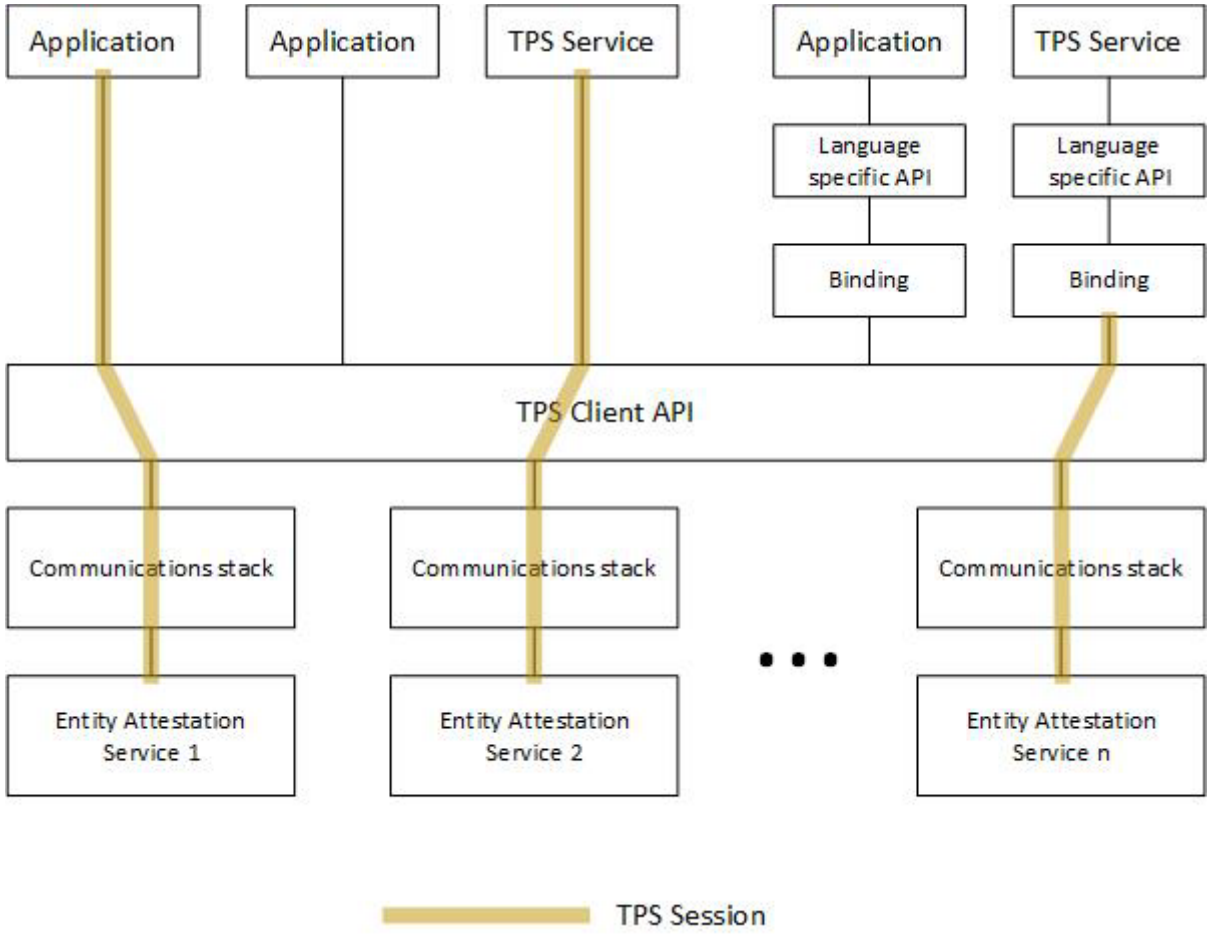
The Entity Attestation API provides means for TPS Clients (i.e. Applications or other TPS Services) to communicate with an Entity Attestation Service and utilize the functionalities it can provide. Entity Attestation Protocol messages are conveyed between a TPS Client and an Entity Attestation Service using the TPS Client API.

An Entity Attestation Service can be implemented on any entity which needs to be able to provide evidence of its nature, status, or condition in order for a client to be able to assess its trustworthiness. In general, the more secure the entity the higher the degree of confidence may be in assessing the evidence. This specification does not limit the environments where an Entity Attestation Service can be implemented.

4.2 Architecture

The following figure outlines the relationship between Entity Attestation related components.

Figure 4-2: Entity Attestation Architecture



TPS Client API is the component used to establish a TPS Session between a TPS Client (an Application or another TPS Service) and an Entity Attestation Service, and to exchange Entity Attestation Protocol messages through the TPS Session. The TPS Session can be viewed as a connection or a channel between the TPS Client and the Entity Attestation Service through which a set of attestation operations can be executed.

An Entity Attestation Service can be accessed using a Language Specific API. In this case, a TPS Client uses the Language Specific API to use the attestation services provided by the Entity Attestation Service. The functions and methods of the Language Specific API are mapped to the Entity Attestation Protocol by a binding. The binding is responsible for converting function and method calls to Entity Attestation Protocol message requests and sending them to the Entity Attestation Service using the TPS Client API. It is also responsible for decoding the incoming Entity Attestation Protocol message response coming from the Entity Attestation Service via the TPS Client API, and constructing the return parameters and values for function and method calls of the Language Specific API.

A device may have more than one Entity Attestation Service available. The TPS Client or the binding is responsible for discovering the required Entity Attestation Service.

4.3 Security

This specification does not impose any security requirements on an environment that implements an Entity Attestation Service. Environments from tamper-resistant Secure Elements to Trusted Execution Environments to Regular Operating Systems can implement an Entity Attestation Service. A given Entity Attestation Parcel might contain nested parcels from different environments, potentially with very different security levels. The environment does need to be a Root of Trust with respect to what is being attested. A Relying Party needs to take into account the nature of the environment that generated an Entity Attestation Parcel when assessing the trustworthiness of the evidence contained within the claims. (Section 5.3 provides an example of how to get information about the nature of the environment.)

4.4 Message Encoding

Entity Attestation Protocol messages are CBOR encoded. A message from a client to a service is termed a request, and a message from the service back to the client is termed a response. Each message takes the form of an Entity Attestation Parcel, with an additional tag that indicates the request or response code. The structure of an Entity Attestation Parcel is described in detail in later sections.

This specification does not place any serialization restrictions on the CBOR used to form the Entity Attestation Parcel, although because Entity Attestation Services may be implemented on resource-limited devices, it is recommended that minimum sizes are used for values, and that indefinite lengths are avoided where possible. Configuration documents can impose restrictions on the serialization to be used.

4.5 Entity Attestation Session

An Entity Attestation Session is a session between a TPS Client and an Entity Attestation Service. It is initiated when the TPS Client opens a TPS Session with the Entity Attestation Service using the `TPSC_SessionOpen` function of the TPS Client API. It is destroyed when the TPS Session with the Entity Attestation Service is closed either by the TPS Client using the `TPSC_SessionClose` function of the TPS Client API, or when the TPS Communication Stack or TPS Client API determines that the TPS Client is no longer active.

When an Entity Attestation Session is closed to a TPS Client, any pending operations related to this TPS Client are abandoned in the Entity Attestation Service.

4.6 Entity Attestation Transaction

An Entity Attestation Transaction is a combination of a request message and a response message which achieves an Entity Attestation Operation. The request message is always sent first, and is only sent by a TPS Client. The request message is always followed by a response message, which is only sent by the Entity Attestation Service.

4.7 Entity Attestation Messages

As with all TPS defined messaging, messages are tagged such that the lowest four digits are the message number and higher digits indicate the specification number.

In this specification, message tags 60000 to 60999 are reserved. Message tags in the range 60000 to 60899 are for GlobalPlatform defined messages. Message tags in the range 60900 to 60999 can be used for proprietary extensions to the GlobalPlatform Entity Attestation Protocol. API users are expected to take appropriate measures to ensure that proprietary extensions are used only when supported, and to be aware that different implementations may have different interpretation of the meaning of a given tag. Proprietary implementations SHALL NOT use any tag reserved to GlobalPlatform use.

4.7.1 Parameters

The following table lists all parameters defined by this specification that can be present in Entity Attestation Protocol messages.

Table 4-1: Entity Attestation Parameters and Assigned Values

Name	Value	Description
parcel	-1	Entity Attestation Parcel
status	-30	Status of an operation

The “`parcel`” parameter contains an Entity Attestation Parcel.

The “`status`” parameter contains an integer value that indicates the result of the attempted operation.

Table 4-2: Status Values and Meanings

Name	Value	Description
`SUCCESS`	0	The operation requested in the corresponding request was successful.
`IO_ERROR`	-1	The operation failed due to an unspecified I/O error.
`INVALID_ARGUMENT`	-3	One or more arguments in the request are invalid.
`GENERAL_FAILURE`	-254	The Entity Attestation Service has suffered a general failure.

4.7.2 TPSEAP_Attest

Description

Informs the Entity Attestation Service that it is to perform an attestation operation.

Request

```
TPSEAP_AttestRequest = #6.60000 ({
-1 => parcel
})
```

Request Parameters

- `parcel`: The request Entity Attestation Parcel which can be used to provide information to the Entity Attestation Service about evidence it should include in the response

Response

```
TPSEAP_AttestResponse = #6.60001 ({
-1 => parcel,
-30 => status
})
```

Response Parameters

- `parcel`: The response Entity Attestation Parcel which can contain evidence
- `status`: Indication of whether the requested operation was successful

Implementation Notes

The status field is used to indicate problems with a request itself, for example that the Entity Attestation Parcel in the request is invalid. When the status is anything other than SUCCESS, the Relying Party cannot rely on any information provided in the parcel parameter.

Responses with status SUCCESS can still include information about problems with parts of the request in the values it returns in the parcel parameter. For example, requested claims that are unrecognized might be omitted from the response.

.

5 OPERATIONS

An Operation is a defined mechanism by which an Entity Attestation Client can obtain attestation information from an Entity Attestation Service. An Operation takes the form of challenge and response. Formally, because an EAP is constructed, transported, and processed in each direction, an Operation is comprised of a Remote Attestation from Client to Service, followed by a Remote Attestation from Service to Client.

For brevity, the term “empty EAP” is used to mean an EAP with an empty Claims Set. Note that it might still be signed, encrypted, or MACed.

5.1 Outline of Parcel Exchange Operations

Every Operation consists of the following steps:

- An Entity Attestation Client creates a Request EAP and sends it to an Entity Attestation Service.
- The Entity Attestation Service parses the Request EAP, constructs a Response EAP, and returns it to the Entity Attestation Client.

If an Operation takes place in a secure channel, it might be that no further cryptographic protection is needed on the EAPs being exchanged, although of course it is still possible to sign, encrypt, or MAC Parcels even in a secure channel.

It is possible to proceed with a more complex attestation procedure in a series of individual steps, for example sending a request for a default Parcel, followed by one or more requests for additional information.

Note that there might be devices which are only capable of returning a fixed Response EAP. However, to promote interoperability for a specific use case, configuration documents could be created that impose requirements for the support of certain Operations.

5.2 Parcel Exchange

The Relying Party performs the following steps to construct a Request EAP.

- Construct a request Claims Set (which can be empty).
- Perform one of the following steps to create a Request Entity Attestation Parcel:
  - Add a tag to the Claims Set to create a Request Unendorsed Claims Set
  - Sign, encrypt, or MAC the Claims Set, and optionally add a tag to create a Request Entity Attestation Token

The Request EAP is sent to the Entity Attestation Service.

On receipt of a Request EAP, the EAS performs the following steps to construct a Response EAP:

- Check the tag values if present.
  - If a tag value is not recognized, return an empty EAP with a suitable error code.
  - Otherwise, interpret the content following the tags according to whether it represents an Entity Attestation Token or an Unendorsed Claims Set.
- If the content is an Entity Attestation Token:
  - If the EAT is MACed, attempt to authenticate the Message Authentication Code, and return an empty EAP with a suitable error code if unable to successfully authenticate the Request EAT.

- If the EAT is encrypted, attempt to decrypt it, and return an empty EAP with a suitable error code if unable to successfully decrypt the Request EAT or interpret the plaintext.
- If the EAT is signed, attempt to verify all signatures present, and return an empty EAP with a suitable error code if unable to successfully verify the Request EAT.

- If the content is an Unendorsed Claims Set:
  - No cryptographic operations are necessary.
- Parse the request Claims Set.
- Retrieve a suitable Response EAP if available.
- Otherwise construct a Response EAP containing a response Claims Set which may consider information in the request Claims Set (especially for Claims that have privacy implications).
- Optionally sign and/or encrypt and/or MAC the Response EAP.
- Optionally tag the (optionally signed, optionally encrypted, optionally MACed) Response EAP.

The Response EAP is returned to the Relying Party.

5.3 Request EAP Claims

In some cases, such as where it is known that the Response EAP is fixed, there is no need for the Request EAP to contain additional information. However, there are cases where the Relying Party provides parameters that influence the construction of the Claims Set in the Response EAP. Claims sent in the Request EAP have specific meanings, defined below. Any other Claim in the Request EAP SHOULD be ignored by the Entity Attestation Service.

5.3.1 Nonce Request Claim

A Nonce Claim included in the Request EAP SHALL include one or more octets. The length and value of the octets is outside the scope of this specification. The Response EAP SHALL contain a Nonce Claim that includes the same length and value of octets in the Request EAP.

This can be used to ensure freshness, in that the Response EAP was generated in response to the Request EAP.

5.3.2 Context Request Claim

A Context Claim included in the Request EAP SHALL include one or more octets. The length, meaning, and value of the octets is outside the scope of this specification. The Response EAP SHALL contain a Context Claim that includes the same length and value of octets in the Request EAP.

This can be used to provide a reference for the Entity Attestation Service that allows it to understand what is making the attestation request, and also to assist the Relying Party in linking a Request / Response EAP pair.

5.3.3 Justification Request Claim

A Justification Claim included in the Request EAP SHALL include a CBOR Byte String. The length, meaning and value of the octets is outside the scope of this specification. The Response EAP SHOULD not contain a Justification Claim.

This can be used to provide information to the Entity Attestation Service about the reason why the Relying Party is entitled to receive some or all the evidence in the Response EAP. For example, it could include proof of possession of a secret value known only to the two parties involved in the Operation.

5.3.4 Important Request Claim

An Important Claim included in the Request EAP SHALL include an array containing one or more Claim Prompts. Each element in the array represents a hint to the Entity Attestation Service about specific Claims which the Relying Party wants to see returned. The Entity Attestation Service SHOULD include the claims matching these hints when constructing the Response EAP.

For various reasons, such as privacy concerns, or uncertainty about the source of the Request EAP, the EAS MAY choose not to include the Claim, or MAY choose to include the Claim without the evidence. If the Relying Party is not provided with evidence it identified as important it might lower its assessment of trustworthiness. However, it could also attempt to retrieve the information another way, for example by sending a further Request EAP with a Claims Set that includes a suitable Justification Claim.

5.3.5 Unnecessary Request Claim

An Unnecessary Claim included in the Request EAP SHALL include an array containing one or more Claim Prompts. Each element in the array represents a hint to the Entity Attestation Service about specific Claims which the Relying Party does not need to see returned. The Entity Attestation Service SHOULD NOT include the claims matching these hints when constructing the Response EAP.

For various reasons, the EAS MAY choose to include the Claim including its evidence, but the Relying Party might not use this evidence in its assessment of trustworthiness.

6 CLAIMS

This specification defines all Claims defined by GlobalPlatform, but imposes no requirements on which Claims need to be supported in any given system.

Certain Claims are identified as “Subclaims”. A Subclaim has an identical format to other Claims, but has a defined meaning only when included in the value field of the parent Claim.

Unless restricted by other documents, any claim can be included in any type of Entity Attestation Parcel, including both an Unendorsed Claims Set and an Entity Attestation Token.

6.1 Data Types

6.1.1 Binary Coded Decimal

A Binary Coded Decimal is a number encapsulated in the smallest possible CBOR Byte String (as discussed in [RFC 8949]) using a BCD format where each digit of the integer is encoded in four bits, with these bits taking values from 0 to 9, from the most significant digit to the least. A single decimal point MAY be encoded in a given number using the decimal value 15. It is an error if the value 15 occurs more than once.

A number containing an odd number of digits (or an even number with a decimal point) SHALL be left justified in the CBOR Byte String, with the most significant digit set to zero. While this encoding is less efficient than some numeric codings, the loss of efficiency may be acceptable for certain types of value.

Examples: 1234567 has the CBOR encoding 4401234567, and 123.4567 has the CBOR encoding 44123F4567.

```
gp_bcd_type = bstr
```

6.1.2 UUID

UUIDs [RFC 4122] are often used as identifiers that are expected to be unique within a given Entity.

UUIDs are encoded as CBOR Byte Strings with a fixed length constraint.

```
uuid = bstr .size 16
```

6.1.3 Claim Prompt

A Claim Prompt is a data type that can reference any Claim Identifier or Subclaim Identifier, whether at the current level of a Claims Set or at a lower level of hierarchy. It is used to provide hints to the Entity Attestation Service about what particular evidence a Relying Party considers important when evaluating trustworthiness.

A Claim Prompt takes one of the following forms:

- Simple
- Subclaims
- Hierarchical

```
gp_claim_prompt_type =
    gp_simple_claim_prompt_type /
    gp_subclaim_claim_prompt_type /
    gp_hierarchical_claim_prompt_type
```

6.1.3.1 Simple Claim Prompt

To prompt for a simple Claim at the current level of hierarchy, the Claim Identifier is included as a CBOR Unsigned Integer.

```
gp_simple_claim_prompt_type = uint
```

6.1.3.2 Subclaim Claim Prompt

To prompt for Subclaims within a Claim at the current level of hierarchy, a CBOR Array with two entries is used. The first entry is a CBOR Unsigned Integer with the value of the Claim Identifier, and the second entry is a nested CBOR Array with zero or more entries.

Each entry in the nested array is a CBOR Unsigned Integer with the value of the Subclaim identifier. An empty array of Subclaim identifiers SHOULD be interpreted to mean a prompt for all Subclaims.

```
gp_subclaim_claim_prompt_type = [ uint, [ * uint ] ]
```

6.1.3.3 Hierarchical Claim Prompt

To prompt for Claims at a lower level of hierarchy, a CBOR Map with a single map pair is used. The map key is a CBOR Unsigned Integer with the value of the Claim Identifier, and the map value is a nested CBOR Map with one or more map pairs.

Each map key in the nested map is an identifier for the hierarchical item. The format of the identifier depends on the Claim Identifier. If the Claim Identifier is "`submods`" as defined in [draft-ietf-rats-eat], the identifier is a CBOR Text String. Other Claim Identifiers can use a CBOR Unsigned Integer, a CBOR Byte String, or a CBOR Text String.

Each map value in the nested map is a CBOR Array with one or more elements, each of which is a Claim Prompt. In general, it is unlikely that Hierarchical Claim Prompts will be nested, and configurations might choose to disallow this. However, Simple and Subclaim Claim Prompts can normally be freely used.

```
gp_hierarchical_claim_prompt_type = {
    uint => {
        + gp_hierarchical_claim_identifier_prompt_type => [
            + gp_claim_prompt_type
        ]
    }
}

gp_hierarchical_claim_identifier_prompt_type = uint / bstr / tstr
```

Examples:

- A prompt for the debug status Claim is `263`, with the CBOR encoding:

  `190107`.

- A prompt for the latitude and longitude Subclaims in the location Claim is `[ 264 , [ 1 , 2 ] ]`, with the CBOR encoding:

  `82190108820102`.

- A prompt for the hardware version and location Claims in a submodule "`board`" is `{ 266: { "board" : [ 260 , [ 264 , [ 1 , 2 ] ] ] } }`, with the CBOR encoding:

  `a119010aa165626f617264821901048219010882010 2`.

- A prompt for the lifecycle state and version Subclaims of a TEE Trusted Application with the identifier 24eba45dc1bb43a49d831221677ab639 is
  { 70500: { h'24eba45dc1bb43a49d831221677ab639' : [70501 , 70502] } }, with the CBOR encoding:

  a11a00011364a15024eba45dc1bb43a49d831221677ab639821a000113651a00011366.

- A prompt for the options and SCP03 keys of an SE Secure Channel Protocol '03' is
  { 80400: { 3 : [80401 , 80402] } } with the CBOR encoding:

  a11a00013a10a103821a00013a111a00013a12.

6.1.4 Security Rating

The Security Rating type encodes information about how secure the Entity is. It is an integer value that takes one of the values defined by the GlobalPlatform Security Task Force:

- 0: Unknown
- 5: Basic
- 10: Substantial
- 15: High

The value Unknown can be used if a value is not relevant in a given context. The Claim Value SHALL be represented as a CBOR Unsigned Integer containing one of the values defined above. All other values are reserved for future use.

```
gp_security_rating_type = &(
    unknown: 0,
    basic: 5,
    substantial: 10,
    high: 15
)
```

6.1.5 SE Secure Element Type

SE Secure Element Type type defines the form factor of the Secure Element implementation.

It is an integer encoding that takes one of the values:

- 0: SE
- 5: eSE
- 10: iSE

All other values are reserved.

```
gp_se_secure_element_type_type = &(
    se : 0,
    ese : 5,
    ise : 10
```

6.1.6 SE Card Lifecycle State Type

The SE Card Lifecycle State type encodes the lifecycle state of the Secure Element.

It is an integer encoding that takes one of the values:

- 0: OP_READY
- 5: INITIALIZED
- 10: SECURED
- 15: CARD_LOCKED
- 20: TERMINATED

All other values are reserved for future use.

```
gp_se_card_lifecycle_state_type = &(
    op_ready: 0,
    initialized: 5,
    secured: 10,
    card_locked: 15,
    terminated: 20
)
```

6.1.7 SE Supplementary Security Domain Type

The SE Supplementary Security Domains Type allows the Supplementary Security Domains installed on the Secure Element to be encoded.

It is defined as a CBOR Map with zero or more entries. Each map pair SHALL consist of a map key that represents the Supplementary Security Domain ID in the form of a CBOR Byte String, and a map value that represents the Supplementary Security Domain parameters in the form of a CBOR Map with zero or more entries, each of which is one of the identified SE SSD Subclaims.

```
gp_se_supplementary_security_domains_type = {
   * gp_se_supplementary_security_domain_id => { * $$GP-SSD-Claims }
}

$$GP-SSD-Claims //= ( &( issuer_number: 1 ) => gp_bcd_type )
$$GP-SSD-Claims //= ( &( image_number: 2 ) => bstr )
$$GP-SSD-Claims //= ( &( key_derivation_data: 3  ) => bstr )
$$GP-SSD-Claims //= ( &( key_information_data: 4  ) => bstr )
$$GP-SSD-Claims //= ( &( default_kvn: 5  ) => uint )
$$GP-SSD-Claims //= ( &( default_kvn_sequence_ctr: 6  ) => bstr )

gp_se_supplementary_security_domain_id = bstr
```

6.1.7.1 issuer_number

The SE Supplementary Security Domain Issuer Number (SIN) Subclaim uniquely identifies the SSD Issuer according to ISO/IEC 7812-1 [ISO 7812-1]. The SIN is of variable length in general, but as used by GlobalPlatform it is limited to either three or four digits.

The Subclaim value SHALL be represented as a CBOR Byte String containing the octets comprising the SIN, in the Binary Coded Decimal form defined in section 6.1.1.

6.1.7.2 image_number

The SE Supplementary Security Domain Image Number (SDIN) Subclaim uniquely identifies an SSD created by a given SSD Issuer. The pair (SIN, SDIN) form a unique identifier for a given SSD. The SDIN is of variable length.

The Subclaim value SHALL be represented as a CBOR Byte String containing the octets comprising the SDIN.

6.1.7.3 key_derivation_data

The SE Supplementary Security Domain Key Derivation Data Subclaim contains key derivation data for the Security Domain.

The Subclaim Value SHALL be represented as a CBOR Byte String containing the key derivation data.

6.1.7.4 key_information_data

The SE Supplementary Security Domain Key Information Data Subclaim contains key information data for the Security Domain.

The Subclaim Value SHALL be represented as a CBOR Byte String containing the key information data.

6.1.7.5 default_kvn

The SE Supplementary Security Domain Default KVN Subclaim contains the default KVN in the Security Domain.

The Subclaim Value SHALL be represented as a CBOR Unsigned Integer containing the default KVN.

6.1.7.6 default_kvn_sequence_ctr

The SE Supplementary Security Domain Default KVN Sequence Counter Subclaim contains the sequence counter for the default KVN in the Security Domain.

The Subclaim Value SHALL be represented as a CBOR Byte String containing the default KVN sequence counter.

6.1.8 TEE Lifecycle State Type

The TEE Lifecycle State Type gives an indication of the lifecycle state of the TEE (as defined by [TMF] `Tee::state`).

It uses an integer encoding that takes one of the values:

- 0: TEE_LOCKED
- 1: TEE_SECURED

All other values are reserved for future use.

```
gp_tee_lifecycle_state_type = &(
    tee_locked: 0,
    tee_secured: 1
)
```

6.1.9 TEE Trusted OS Architectures Type

The TEE Trusted OS Architectures Type defines how to encode details of instruction sets and architectures which can be used by Trusted Applications running in the TEE.

It represents data as a CBOR Map with zero or more entries. Each map pair ([TMF] `ISA`) SHALL consist of a map key that represents the Instruction Set Architecture Name in the form of a CBOR Text String, and a map value that represents the TEE Trusted OS Architecture parameters in the form of a CBOR Map with zero or more entries, each of which is one of the identified TEE Trusted OS ISA Subclaims.

```
gp_tee_trusted_os_architectures_type = {
  * &(arch_name: tstr) => {
      ? &( isa_processor_type: 0) => tstr,
      ? &( isa_instruction_set: 1) => tstr,
      ? &( isa_address_size: 2) => uint,
      ? &( isa_abi_information: 3) => uint,
      ? &( isa_endianness: 4) => gp_endianness_type
  }
}
```

Note: Endianness is encoded using Endianness, defined in section 38.

6.1.9.1 isa_processor_type

The ISA Processor Type Subclaim indicates the type of the processor ([TMF] `ISA::processorType`).

The Subclaim Value SHALL be represented as a CBOR Text String containing the processor type.

6.1.9.2 isa_instruction_set

The ISA Instruction Set Subclaim specifies the instruction set as a string ([TMF] `ISA::instructionSet`).

The Subclaim Value SHALL be represented as a CBOR Text String containing the instruction set description.

6.1.9.3 isa_address_Size

The ISA Address Size defines the size of addresses in bits ([TMF] `ISA::addressSize`).

The Subclaim Value SHALL be represented as a CBOR Unsigned Integer containing the address size in bits.

6.1.9.4 isa_abi_information

The SA Application Binary Interface Information Claim specifies the ABI that is in use ([TMF] `ISA::abi`).

The Subclaim Value SHALL be represented as a CBOR Text String containing information about the ABI.

6.1.10 isa_endianness

The Endianness Type specifies how values greater than 1 byte in length are stored ([TMF] `ISA::endianness`).

It is an integer encoding that takes one of the values:

- 0: Little Endian
- 1: Big Endian
- 2..127: Reserved
- 128..255: Implementation dependent

```
gp_endianness_type = &(
    little_endian: 0,
    big_endian: 1,
    other_endian: 2..255
)
```

6.1.11 TEE Options Type

The Options Type allows list of options, in the form of pairs of `(name, integer)`, to be encoded.

Options SHALL be encoded on a CBOR Map with zero or more entries, each of which SHALL be a CBOR Text String containing the name (key) and a CBOR Unsigned Integer containing the version (value).

```
gp_tee_options_type = {
     * tstr => uint
}
```

6.1.12 TEE Implementation Properties Type

The TEE Implementation Properties Type allows TEE properties to be encoded. There is a set of defined keys for standardized properties and the ability to encode implementation-defined properties.

TEE Implementation Properties SHALL be represented as a CBOR Map with zero or more entries. Each entry is identified by an integer key. Unused positive integer values are reserved for future versions of this specification. Negative keys can be used for implementation-specific properties.

Note: The key values given below are informative only. The normative source for TEE property identifier keys is [TEE Core], section 4.7, and keys defined therein MUST be used in the event of conflict with this specification.

```
gp_tee_implementation_properties_type = {
  ? &( gpd_tee_apiversion: 0  ) => tstr,
  ? &( gpd_tee_internalCore_version: 1  ) => int,
  ? &( gpd_tee_description: 2  ) => tstr,
  ? &( gpd_tee_deviceID: 3  ) => uuid,
  ? &( gpd_tee_name: 4  ) => tstr,
  ? &( gpd_tee_implementationstackhash: 5  ) => bstr,
  ? &( gpd_tee_implementationstack: 6  ) => bstr .cbor
gp_tee_globalplatform_component
,
  ? &( gpd_tee_systemTime_protectionLevel: 7  ) => int,
  ? &( gpd_tee_TAPersistentTime_protectionLevel: 8  ) => int,
  ? &( gpd_tee_arith_maxBigIntSize: 9  ) => int,
  ? &( gpd_tee_cryptography_ecc: 10  ) => bool,
  ? &( gpd_tee_cryptography_nist: 11  ) => bool,
  ? &( gpd_tee_cryptography_bsi_r: 12  ) => bool,
  ? &( gpd_tee_cryptography_bsi_t: 13  ) => bool,
  ? &( gpd_tee_cryptography_ietf: 14  ) => bool,
  ? &( gpd_tee_cryptography_octa: 15  ) => bool,
  ? &( gpd_tee_cryptography_sec: 16  ) => bool,
  ? &( gpd_tee_cryptography_NISTpqc_crystals: 17  ) => bool,
  ? &( gpd_tee_cryptography_NISTpqc_SLHDSA: 19  ) => bool,
  ? &( gpd_tee_cryptography_statefulverification: 20  ) => bool,
  ? &( gpd_tee_cryptography_statefulsignatures: 21  ) => bool,
  ? &( gpd_tee_cryptography_FrodoKEM: 22  ) => bool,
  ? &( gpd_tee_cryptography_NTRU: 23  ) => bool,
  ? &( gpd_tee_cryptography_maxretaineddata: 25 ) => int,
  ? &( gpd_tee_trustedStorage_private_rollbackProtection: 26  ) => int,
  ? &( gpd_tee_trustedStorage_perso_rollbackProtection: 27  ) => int,
  ? &( gpd_tee_trustedStorage_protected_rollbackProtection: 28  ) => int,
  ? &( gpd_tee_trustedStorage_antiRollback_protectionLevel: 29  ) => int,
  ? &( gpd_tee_trustedStorage_rollbackDetection_protectionLevel: 30  ) =>
int,
  ? &( gpd_tee_trustedos_implementation_version: 31  ) => tstr,
  ? &( gpd_tee_trustedos_implementation_binaryversion: 32  ) => bstr,
  ? &( gpd_tee_trustedos_manufacturer: 33  ) => tstr,
  ? &( gpd_tee_firmware_implementation_version: 34  ) => tstr,
  ? &( gpd_tee_firmware_implementation_binaryversion: 35  ) => bstr,
  ? &( gpd_tee_firmware_manufacturer: 36  ) => tstr,
  ? &( gpd_tee_event_maxSources: 37  ) => int,
  ? &( gpd_tee_maskState: 38  ) => bool,
  * uint .gt 38 => any,
  * nint => gp_tee_properties_value
}
```

References to property definitions below:

- `gpd_tee_apiversion`: defined as `gpd.tee.apiversion` in [TEE Core] section 4.7.
- `gpd_tee_internalCore_version`: defined as `gpd.tee.internalCore.version` in [TEE Core] section 4.7.
- `gpd_tee_description`: defined as `gpd.tee.description` in [TEE Core] section 4.7.
- `gpd_tee_deviceID`: defined as `gpd.tee.deviceID` in [TEE Core] section 4.7.
- `gpd_tee_name`: defined as `gpd.tee.name` in [TEE Core] section 4.7.
- `gpd_tee_implementationstackhash`: defined as `gpd.tee.implementationstackhash` in [TEE Core] section 4.7.
- `gpd_tee_implementationstack`: defined as `gpd.tee.implementationstack` in [TEE Core] section 4.7.
- `gpd_tee_systemTime_protectionLevel`: defined as `gpd.tee.systemTime.protectionLevel` in [TEE Core] section 4.7.
- `gpd_tee_TAPersistentTime_protectionLevel`: defined as `gpd.tee.TAPersistentTime.protectionLevel` in [TEE Core] section 4.7.
- `gpd_tee_arith_maxBigIntSize`: defined as `gpd.tee.arith.maxBigIntSize` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_ecc`: defined as `gpd.tee.cryptography.ecc` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_nist`: defined as `gpd.tee.cryptography.nist` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_bsi_r`: defined as `gpd.tee.cryptography.bsi-r` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_bsi_t`: defined as `gpd.tee.cryptography.bsi_t` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_ietf`: defined as `gpd.tee.cryptography.ietf` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_octa`: defined as `gpd.tee.cryptography.octa` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_sec`: defined as `gpd.tee.cryptography.sec` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_NISTpqc_crystals`: defined as `gpd.tee.cryptography.NISTpqc.crystals` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_NISTpqc_SLHDSA`: defined as `gpd.tee.cryptography.NISTpqc.SLHDSA` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_statefulverification`: defined as `gpd.tee.cryptography.statefulverification` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_statefulsignatures`: defined as `gpd.tee.cryptography.statefulsignatures` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_FrodoKEM`: defined as `gpd.tee.cryptography.FrodoKEM` in [TEE Core] section 4.7.
- `gpd_tee_cryptography_NTRU`: defined as `gpd.tee.cryptography.NTRU` in [TEE Core] section 4.7.

- `gpd_tee_cryptography_maxretaineddata`: defined as `gpd.tee.cryptography.maxretaineddata` in [TEE Core] section 4.7.
- `gpd_tee_trustedStorage_private_rollbackProtection`: defined as `gpd.tee.trustedStorage.private.rollbackProtection` in [TEE Core] section 4.7.
- `gpd_tee_trustedStorage_perso_rollbackProtection`: defined as `gpd.tee.trustedStorage.perso.rollbackProtection` in [TEE Core] section 4.7.
- `gpd_tee_trustedStorage_protected_rollbackProtection`: defined as `gpd.tee.trustedStorage.protected.rollbackProtection` in [TEE Core] section 4.7.
- `gpd_tee_trustedStorage_antiRollback_protectionLevel`: defined as `gpd.tee.trustedStorage.antiRollback.protectionLevel` in [TEE Core] section 4.7.
- `gpd_tee_trustedStorage_rollbackDetection_protectionLevel`: defined as `gpd.tee.trustedStorage.rollbackProtection.protectionLevel` in [TEE Core] section 4.7.
- `gpd_tee_trustedos_implementation_version`: defined as `gpd.tee.trustedos.implementation.version` in [TEE Core] section 4.7.
- `gpd_tee_trustedos_implementation_binaryversion`: defined as `gpd.tee.trustedos.implementation.binaryversion` in [TEE Core] section 4.7.
- `gpd_tee_trustedos_manufacturer`: defined as `gpd.tee.trustedos.manufacturer` in [TEE Core] section 4.7.
- `gpd_tee_firmware_implementation_version`: defined as `gpd.tee.firmware.implementation.version` in [TEE Core] section 4.7.
- `gpd_tee_firmware_implementation_binaryversion`: defined as `gpd.tee.firmware.implementation.binaryversion` in [TEE Core] section 4.7.
- `gpd_tee_firmware_manufacturer`: defined as `gpd.tee.firmware.manufacturer` in [TEE Core] section 4.7.
- `gpd_tee_event_maxsources`: defined as `gpd.tee.event.maxSources` in [TEE Core] section 4.7.
- `gpd_tee_maskState`: defined as `gpd.tee.maskState` in [TEE Core] section 4.7.

Implementation-defined properties can be represented using the TEE Properties Value type defined in section 6.1.13.

6.1.13 TEE Properties Value Type

TEE Properties Value Type allows implementation-specific properties to be encoded. [TEE Core] section 4.4 provides normative information on property encoding.

```
gp_tee_properties_value = (bool
    / uint
    / tstr
    / bstr
    / gp_tee_identity)
```

6.1.14 TEE Identity Type

TEE Identity Type allows a `TEE_Identity` to be encoded. [TEE Core], sections 4.1.1 and 4.4 provide normative information on `TEE_Identity`.

```
gp_tee_identity = {
  &( loginMethod: 0  ) => uint,
  &( uuid: 1  ) => uuid
}
```

6.1.15 TEE Trusted Applications Type

TEE Trusted Applications Type allows information about Trusted Applications hosted by a TEE to be encoded.

It is defined as a CBOR Map with zero of more entries. Each map pair ([TMF] `TrustedApplication`) SHALL consist of a map key that represents the Trusted Application ID in the form of a CBOR Byte String, and a map value that represents the Trusted Application parameters in the form of a CBOR Map with zero or more entries, each of which is one of the identified TEE TA Subclaims.

```
gp_tee_trusted_applications_type = { * &(application_)id: bstr) => {
    ? &( parent: 0)            => uuid,
    ? &( lifecycle_state: 1 )  => gp_tee_ta_lifecycle_state_type,
    ? &( version: 2 )          => tstr,
    ? &( measurement: 3 )      => bstr,
    ? &( description: 4 )      => tstr,
    ? &( properties: 5 )       => { * tstr => gp_tee_properties_value },
    ? &( extra_data: 6 )       => bstr
  }
}
```

6.1.15.1 parent

The `parent` Subclaim provides the UUID of the parent Security Domain ([TMF] `TrustedApplication::parent`).

The Subclaim Value SHALL be represented as a CBOR Byte String containing the UUID of the parent Security Domain.

6.1.15.2 lifecycle_state

The `lifecycle_state` Subclaim is encoded as defined in section 6.1.16.

6.1.15.3 version

The `version` Subclaim provides the version of the Trusted Application ([TMF] `TrustedApplication::version`).

The Subclaim Value SHALL be represented as a CBOR Text String containing the version of the Trusted Application.

6.1.15.4 measurement

The `measurement` Subclaim provides a measurement for the TA.

> **Note:** This specification leaves the mechanism used to generate a measurement to the implementation. Examples of possible measurements include but are not limited to: hash of the TA binary; DICE-based chained measurement value.

6.1.15.5 description

The `description` Subclaim provides an informative text description of the TA. The format of the contents of this description are defined by the TA implementation and outside of the scope of this specification.

6.1.15.6 properties

The properties Subclaim allows TA properties to be encoded using a CBOR Map where the keys are property names (as CBOR String types) and the values are encoded using the TEE Properties Value Type (section 6.1.13).

6.1.15.7 extra_data

The `extra_data` Subclaim allows implementation-specific information about a TA to be encoded as a CBOR `bstr`. This specification does not further specify the format of the contents of the `bstr`.

6.1.16 TEE Trusted Application Lifecycle State Type

The TEE Trusted Application Lifecycle State Type encodes the lifecycle state of the Trusted Application ([TMF] `TrustedApplication::lifecycleState`).

It is an integer encoding that takes one of the values:

- 0: `taInactiveState`
- 1: `taExecutableState`
- 2: `taLockedState`

All other values are reserved for future use.

```
gp_tee_ta_lifecycle_state_type = &(
    ta_inactive: 0,
    ta_executable : 1,
    ta_locked : 2
)
```

6.1.17 TEE Security Domains Type

The TEE Security Domains Type encodes the set of Security Domains present in the TEE.

The Security Domain set SHALL be represented as a CBOR Map with zero or more entries. Each map pair ([TMF] `SecurityDomain`) SHALL consist of a map key that represents the Security Domain ID in the form of a CBOR Byte String, and a map value that represents the Security Domain parameters in the form of a CBOR Map with zero or more entries, each of which is one of the identified TEE SD Subclaims.

```
gp_tee_security_domains_type = {
    * gp_tee_security_domain_id => {
        ? &( sd_parent: 0 )           => uuid,
        ? &( sd_lifecycle_state: 1 ) => gp_tee_sd_lifecycle_state_type,
        ? &( sd_authority: 2 )        => gp_tee_sd_authority_type,
        ? &( sd_privileges: 3 )       => gp_tee_sd_privileges_type,
        ? &( sd_protocols: 4 )        => gp_tee_sd_protocols_type
    }
}

gp_tee_security_domain_id = bstr
```

6.1.17.1 sd_parent

The TEE Security Domain Parent Subclaim provides the UUID of the parent Security Domain ([TMF] `SecurityDomain::parent`) if applicable.

The Subclaim Value SHALL be represented as a `uuid` type encoding the UUID of the parent Security Domain.

6.1.17.2 sd_lifecycle_state

The TEE Security Domain Lifecycle State Subclaim provides the lifecycle state of the Security Domain ([TMF] `SecurityDomain::lifecycleState`).

The Subclaim Value SHALL be represented as a TEE Security Domain Lifecycle State Type (section 6.1.18)

6.1.17.3 sd_authority

The TEE Security Domain Authority Subclaim provides additional information about the Authority that manages the Security Domain ([TMF] `SecurityDomain::authority`).

The Subclaim Value SHALL be represented TEE Security Domain Authority Type (section 6.1.19).

6.1.17.4 sd_privileges

The TEE Security Domain Privileges Subclaim contains the privileges of the Security Domain ([TMF] `SecurityDomain::privileges`).

The Subclaim SHALL be encoded using a TEE Security Domain Privileges Type (section 6.1.20).

6.1.17.5 sd_protocols

The TEE Security Domain ID Subclaim contains a list of protocols supported by the Security Domain related to the Security Layer implementation ([TMF] `SecurityDomain::protocols`).

The Subclaim SHALL be encoded using a TEE Security Domain Protocols Type (section 6.1.21).

6.1.18 TEE Security Domain Lifecycle State Type

The TEE Security Domain Lifecycle State Type encodes the lifecycle state of the Security Domain ([TMF] `SecurityDomain::lifecycleState`).

It is a CBOR integer that takes one of the values:

- 0: `sdBlockedState`
- 1: `sdActiveState`
- 2: `sdRestrictedState`

All other values are reserved for future use.

```
gp_tee_sd_lifecycle_state_type = &(
    sd_blocked: 0,
    sd_active: 1,
    sd_restricted: 2
)
```

6.1.19 TEE Security Domain Authority Type

The TEE Security Domain Authority Type encodes additional information about the Authority that manages the Security Domain ([TMF] `SecurityDomain::authority`).

It is a CBOR Map containing either one or two entries.

- The first entry SHALL have a key of 1 and a value of type CBOR Text String containing the Authority name.
- If present, the second entry SHALL have a key of 2 and a value of type CBOR Text String containing a URL for the Authority.

```
gp_tee_sd_authority_type = {
    &( name: 0  )    => tstr,
  ? &( urlInfo: 1  ) => tstr
}
```

6.1.20 TEE Security Domain Privileges Type

The TEE Security Domain Privileges Type encodes information about the privileges of the Security Domain ([TMF] `SecurityDomain::privileges`).

The Subclaim Value SHALL be represented as a CBOR Map containing either one or two entries.

- The first entry SHALL have key 0 and value of a CBOR Array containing zero or more entries (value). Each element in the array SHALL be a CBOR Map containing either one or two entries:
  - The first entry SHALL have key 0 and a value of type CBOR Unsigned Integer
  - If present, the second entry SHALL have key 1 and a value of type CBOR Byte String containing privilege parameters (value)
- If the Security Domain is a root SD, a second entry SHALL be included with key 1 and value of type CBOR simple(21) representing “`true`” (value). Otherwise, a second entry SHALL NOT be included.

```
gp_tee_sd_privileges_type = {
    &( listOfPrivileges: 0  ) => [ * {
        &( privilegeID: 0  ) => uint,
      ? &( privilegeParams: 1  ) => bstr
    } ],
  ? &( isRootSD: 1  ) => true
}
```

6.1.21 TEE Security Domain Protocols Type

The TEE Security Domain Protocols Type encodes a set of protocols supported by a Security Domain related to the Security Layer implementation ([TMF] `SecurityDomain::protocols`).

It SHALL be represented as a CBOR Array containing zero or more entries. Each element in the array SHALL be a CBOR Map containing either one or two entries.

- The first entry SHALL have key 0 and value of type UUID representing the protocol UUID (value)
- If present, the second entry SHALL have key 1 and value of type CBOR Byte String containing privilege parameters (value)

```
gp_tee_sd_protocols_type = [ * {
    &( protocols: 0  )    => uuid,
  ? &( protocolInfo: 1  ) => bstr
} ]
```

6.1.22 TEE Client Properties Type

The TEE Client Properties Type encodes information related to the TEE managed by a given instance of the TEE Client API.

It SHALL be represented as a CBOR Map containing one or more entries.

- The first entry SHALL have key 0 and a value of type TEE Identity Type (section 6.1.14) encoding the `TEE_Identity` as specified in [TEE Core].
- If present, the second entry SHALL have key 1 and a value of type `isa_endianness` (section 6.1.10) specifying how values stored over more than one byte are stored by the TEE.
- If present, the third entry SHALL have key 2 and a CBOR byte string containing an encoded `gp_tee_pathList`, as specified in [TEE Core], section 4.7.
- Additional keys and values MAY be present. Positive integer keys are reserved by GlobalPlatform and MUST NOT be used to specify implementation-specific properties. Negative keys MAY be used to specify implementation-specific properties

```
gp_tee_client_properties_type = {
       &( gpd_client_identity: 0  ) => gp_tee_identity,
     ? &( gpd_client_endian: 1  )   => gp_tee_ta_endianness_type,
     ? &( gpd_client_path: 2  )     => bstr .cbor gp_tee_pathList

     ; Positive keys are reserved for future GlobalPlatform use
       uint .gt 3 => any,

     ; Impementation specific properties must have negative keys
       nint => gp_tee_implementation_properties_value
}
```

6.2 Privacy

Some Claims may contain information with privacy implications, for example:

- Fixed device identifiers.
- Signing or encryption key material that is unique to a given device.
- Current device location.

Because what is regarded as private depends on the context, it is not possible to label Claims as “private” or “not private” in this specification. Configuration documents could include requirements on privacy aspects of Claims.

6.3 Claims from the CBOR Web Token Specification

The following Claims are included from the CBOR Web Token specification [RFC 8392], maintaining the same Claim Key, Claim Name, and requirements for the formatting of the Claim Value.

- `iss` (Issuer) Claim
- `sub` (Subject) Claim
- `aud` (Audience) Claim
- `exp` (Expiration Time) Claim
- `nbf` (Not Before) Claim
- `iat` (Issued At) Claim
- `cti` (CWT ID) Claim

The evidence contained in Issuer, Subject, and Audience Claims might contain information that can be used for identification and tracking. As such, care needs to be taken to ensure that the Claim Value in these Claims is compliant with any applicable privacy policies.

6.4 Claims from the Entity Attestation Token Specification

The following Claims are included from the IETF Entity Attestation Token specification [draft-ietf-rats-eat], maintaining the same Claim Key, Claim Name, and requirements for the formatting of the Claim Value.

- EAT Nonce Claim (`eat_nonce`)
- Universal Entity ID Claim (`ueid`)
- Semi-Permanent UEIDs Claim (`sueids`)
- Hardware OEM Identification Claim (`oemid`)
- Hardware Model Claim (`hwmodel`)
- Hardware Version Claim (`hwversion`)
- Software Name Claim (`swname`)
- Software Version Claim (`swversion`)
- OEM Authorized Boot Claim (`oemboot`)
- Debug Status Claim (`dbgstat`)
- Location Claim (`location`)
  - Latitude Sub-Claim (`latitude`)
  - Longitude Sub-Claim (`longitude`)
  - Altitude Sub-Claim (`altitude`)
  - Accuracy Sub-Claim (`accuracy`)
  - Altitude Accuracy Sub-Claim (`altitude-accuracy`)
  - Heading Sub-Claim (`heading`)
  - Speed Sub-Claim (`speed`)
  - Timestamp Sub-Claim (`timestamp`)
  - Age Sub-Claim (`age`)
- Uptime Claim (`uptime`)
- Boot Count Claim (`bootcount`)
- Boot Seed Claim (`bootseed`)
- Digital Letters of Approval Claim (`dloas`)
- Software Manifests Claim (`manifests`)
- Measurements Claim (`measurements`)
- Software Measurement Results Claim (`measres`)
- Submodules (`submods`)
- Timestamp Claim (`iat`)
- EAT Profile Claim (`eat_profile`)
- Intended Use Claim (`intuse`)

[draft-ietf-rats-eat] also includes all the Claims defined in the CBOR Web Token specification [RFC 8392].

The evidence contained in claims such as Location might contain information that can be used for identification and tracking. As such, care needs to be taken to ensure that the Claim Value in these Claims is compliant with any applicable privacy policies.

Because of the possibility of location spoofing, care needs to be taken in evaluating the evidence provided in a Location Claim, even from a trusted source.

Similarly, because insecure clocks can be tampered with, care needs to be taken in evaluating the evidence provided in claims such as “Uptime” or “Issued At”.

6.5 Top Level Claims

Top-level claims are included within the main EAT claims map. Therefore, they MUST be registered with IANA on the CWT registry.

6.5.1 GlobalPlatform Component

The GlobalPlatform Component claim defines a map under which all claims for GlobalPlatform components are managed.

IANA Considerations: The value of `globalplatform_component` MUST be a value assigned on the IANA CBOR Web Token (CWT) Claims registry.

Editor's note: Document cannot be published until IANA has returned a value for `globalplatform_component`. The value <IANA_ASSIGNED> below MUST be replaced with the IANA assigned value. This note can be removed when IANA assignment is complete.

```
$$Claims-Set-Claims //= (globalplatform_component => gp_component_claims)

globalplatform_component = <IANA_ASSIGNED>

gp_component_claims = { * $$GP-Claims }
```

6.6 Claims Applicable to All Entities

6.6.1 Security Rating

The Security Rating Claim provides information about how secure the Entity is.

The claim key SHALL be 100 and the claim value SHALL be of type `gp_security_rating_type` defined in section 6.1.4.

```
$$GP-Claims //= (
    &(gp_security_rating: 100) => gp_security_rating_type
)
```

6.6.2 MUD File URL

The MUD File URL Claim includes information about where a Manufacturer Usage Description (MUD) ([RFC 8520]) file can be found.

The claim key SHALL be 101 and the claim value SHALL be represented as a CBOR Text String.

```
$$GP-Claims //= ( &(gp_mud_file_url: 101) => tstr )
```

6.6.3 Legacy Material

A Claims Set may need to include material that is not defined by this specification. The Legacy Material Claim allows this to be included. The generation and parsing of the content are out of scope of this specification.

The claim key SHALL be 102 and the claim value SHALL be represented as a CBOR Byte String. If the material is in the form of a string of octets, they are placed directly into the payload. If the material is in the form of text, the UTF-8 equivalent is placed into the payload.

```
$$GP-Claims //= ( &(gp_legacy_material: 102) => bstr )
```

6.6.4 Justification

The Justification Claim allows the Relying Party to provide some justification as to why it is entitled to the evidence it needs. The generation and parsing of the content are out of scope of this specification, but for example it might consist of proof of ownership of a shared secret.

The claim key SHALL be 103 and the claim value SHALL be represented as a CBOR Byte String. If the material is in the form of a string of octets, they are placed directly into the payload. If the material is in any other form, the CBOR encoding is placed into the payload.

```
$$GP-Claims //= ( &(gp_justification: 103) => bstr )
```

6.6.5 Context

The Context Claim allows the Relying Party to provide context to be considered when processing a request. The generation and parsing of the content are out of scope of this specification, but for example it might consist of an application identifier.

The claim key SHALL be 104 and the claim value SHALL be represented as a CBOR Byte String. If the material is in the form of a string of octets, they are placed directly into the payload. If the material is in any other form, the CBOR encoding is placed into the payload.

```
$$GP-Claims //= ( &(gp_context: 104) => bstr )
```

6.6.6 Important

The Important Claim allows the Relying Party to provide information about the specific claims or subclaims which it needs to assess trustworthiness. This can assist the Entity Attestation Service in ensuring that relevant information is provided where possible.

The claim key SHALL be 105 and the claim value SHALL be represented as a CBOR Array with zero or more entries, each of which SHALL be of the Claim Prompt data type (section 6.1.2).

```
$$GP-Claims //= ( &(gp_important: 105) => [* gp_claim_prompt_type ] )
```

6.6.7 Unnecessary

The Unnecessary Claim allows the Relying Party to provide information about the specific claims or subclaims which it does not require to assess trustworthiness. This can help the Entity Attestation Service minimize the amount of information it returns.

The claim key SHALL be 106 and the claim value SHALL be represented as a CBOR Array with zero or more entries, each of which SHALL be of the Claim Prompt data type (section 6.1.2).

```
$$GP-Claims //= ( &(gp_unnecessary: 106) => [* gp_claim_prompt_type ] )
```

6.6.8 GP EAT Version

The GP EAT Version claim allows an Attester to indicate the version of this specification that it supports.

The claim key SHALL be 107 and the claim value SHALL be represented as an array containing two, three or four units, where:

- The first entry contains the major version of the specification.
- The second entry contains the minor version of the specification.
- The third entry, if present, contains the update version of the specification.
- The fourth entry, if present, contains the patch number of the specification.

As an example, `[1,3]` represents specification version 1.3 and `[1,0,1,3]` represents specification version 1.0.1.3.

```
$$GP-Claims //= ( &gp_eat_version: 107) => [ 2*4 uint ] )
```

6.7 Claims Applicable to Secure Elements

This section defines Claims which are applicable to Secure Elements conforming to the GlobalPlatform Card Specification [GPCS].

6.7.1 SE ISD Issuer Identification Number

The SE ISD Issuer Identification Number (IIN) Claim uniquely identifies the Card Issuer according to ISO/IEC 7812-1 [ISO 7812-1]. According to [GPCS] section 7.4.1.1, it may be used to associate the card with a particular Card Management System. The IIN is of variable length in general, but as used by GlobalPlatform it is limited to either three or four digits.

The claim key SHALL be 150 and the claim value SHALL be represented as a CBOR Byte String containing the octets comprising the IIN, in the Binary Coded Decimal form defined in section 6.1.1.

```
$$GP-Claims //= ( &(gp_se_isd_issuer_id_number: 150) => gp_bcd_type )
```

6.7.2 SE ISD Card Image Number

The SE ISD Card Image Number (CIN) Claim uniquely identifies a card provided by a given Card Issuer. The pair (IIN, CIN) form a unique identifier for a given Card. According to [GPCS], it may be used to identify an individual Card with a particular Card Management System. CIN is of variable length.

The claim key SHALL be 151 and the claim value SHALL be represented as a CBOR Byte String containing the octets comprising the CIN.

```
$$GP-Claims //= ( &(gp_se_isd_card_image_number: 151) => bstr )
```

6.7.3 SE ISD CRD Runtime Type

The SE ISD CRD Runtime Type Claim contains information from the Card Recognition Data about the type of the runtime environment.

The claim key SHALL be 152 and the claim value SHALL be represented as a CBOR Unsigned Integer containing the relevant code.

```
$$GP-Claims //= ( &(gp_se_isd_crd_runtime_type: 152) => uint )
```

6.7.4 SE ISD CRD Runtime Version

The SE ISD CRD Runtime Version Claim contains information from the Card Recognition Data about the version of the runtime environment.

The claim key SHALL be 153 and the claim value SHALL be represented as a CBOR Byte String containing 3 octets comprising the version number beginning with the major version number octet.

```
$$GP-Claims //= (
    &(gp_se_isd_crd_runtime_version: 153) => bstr .size (3)
)
```

6.7.5 SE ISD CRD GlobalPlatform Configuration Identifier

The SE ISD CRD GlobalPlatform Configuration Identifier Claim contains information from the Card Recognition Data about the identity of the GlobalPlatform Configuration.

The claim key SHALL be 154 and the claim value SHALL be represented as a CBOR Unsigned Integer containing the relevant code.

```
$$GP-Claims //= ( &(gp_se_isd_gp_configuration_id: 154) => uint )
```

6.7.6 SE ISD CRD GlobalPlatform Configuration Version

The SE ISD CRD GlobalPlatform Configuration Version Claim contains information from the Card Recognition Data about the version of the GlobalPlatform Configuration.

The claim key SHALL be 155 and the claim value SHALL be represented as a CBOR Byte String containing 3 octets comprising the version number beginning with the major version number octet.

```
$$GP-Claims //= (
    &(gp_se_isd_crd_gp_configuration_version: 155) => bstr .size (3)
)
```

6.7.7 SE ISD CCI Secure Channel Protocols

The SE ISD CCI Secure Channel Protocols Claim contains information from the Card Capability Information about secure channels that are supported.

The claim key SHALL be 156 and the claim value SHALL be represented as a CBOR Map with zero of more entries. Each map pair SHALL consist of a map key that represents the Secure Channel Protocol Type in the form of a CBOR Unsigned Integer, and a map value that represents the Secure Channel Protocol parameters in the form of a CBOR Map with zero or more entries, each of which is one of the identified SE SCP Subclaims.

```
$$GP-Claims //= (
    &(gp_se_isd_cci_secure_channel_protocols: 156) =>
        gp_se_isd_cci_secure_channel_protocols_type
)

gp_se_isd_cci_secure_channel_protocols_type = {
   * gp_se_isd_cci_secure_channel_protocol_number => { * $$GP-SCP-Claim }
}

$$GP-SCP-Claim //= ( &(gp_se_isd_cci_scp_options: 1) => bstr)
$$GP-SCP-Claim //= ( &(gp_se_isd_cci_scp_scp03_keys: 2) => bstr)
$$GP-SCP-Claim //= ( &(gp_se_isd_cci_scp_scp81_tls_suites : 3) => bstr)
$$GP-SCP-Claim //= ( &(gp_se_isd_cci_scp_scp81_max_length: 4) => uint)

gp_se_isd_cci_secure_channel_protocol_number = uint
```

6.7.7.1 SE ISD CCI Secure Channel Protocol Options

The SE ISD CCI Secure Channel Protocol Options Subclaim contains information about options relevant to the secure channel protocol.

The subclaim value SHALL be represented as a CBOR Byte String containing octets for the options.

6.7.7.2 SE ISD CCI Secure Channel Protocol SCP03 Keys

The SE ISD CCI Secure Channel Protocol SCP03 Keys Subclaim defines the keys supported for Secure Channel Protocol '03'.

The subclaim value SHALL be represented as a CBOR Byte String containing octets for the supported keys.

6.7.7.3 SE ISD CCI Secure Channel Protocol SCP81 TLS Suites

The SE ISD CCI Secure Channel Protocol SCP81 TLS Suites Subclaim defines the TLS cypher suites supported for Secure Channel Protocol '81'.

The subclaim value SHALL be represented as a CBOR Byte String containing octets for the supported TLS cypher suites.

6.7.7.4 SE ISD CCI Secure Channel Protocol SCP81 Max Length

The SE ISD CCI Secure Channel Protocol SCP81 Max Length Subclaim defines the maximum length of pre-shared keys in bytes for Secure Channel Protocol '81'.

The subclaim value SHALL be represented as a CBOR Unsigned Integer containing relevant length in bytes.

6.7.8 SE ISD CCI SSD Privileges

The SE ISD CCI SSD Privileges Claim contains information about privileges that can be assigned to a Supplementary Security Domain.

The claim key SHALL be 157 and the claim value SHALL be represented as a CBOR Byte String containing 3 octets for the privileges that can be assigned.

```
$$GP-Claims //= (
    &(gp_se_isd_cci_ssd_privileges: 157) => bstr .size (3)
)
```

6.7.9 SE ISD CCI Application Privileges

The SE ISD CCI Application Privileges Claim contains information about privileges that can be assigned to an application.

The claim key SHALL be 158 and the claim value SHALL be represented as a CBOR Byte String containing 3 octets for the privileges that can be assigned.

```
$$GP-Claims //= (
    &(gp_se_isd_cci_application_privileges: 158) => bstr .size (3)
)
```

6.7.10 SE ISD CCI LFDB Hash Algorithms

The SE ISD CCI LFDB Hash Algorithms Claim defines the supported Load File Data Block hash algorithms.

The claim key SHALL be 159 and the claim value SHALL be represented as a CBOR Byte String containing octets for the supported algorithms.

```
$$GP-Claims //= ( &(gp_se_isd_cci_lfdb_hash_algorithms: 159) => bstr )
```

6.7.11 SE ISD CCI LFDB Encryption Cypher Suites

The SE ISD CCI LFDB Encryption Cypher Suites Claim defines the supported cypher suites for Load File Data Block encryption.

The claim key SHALL be 160 and the claim value SHALL be represented as a CBOR Byte String containing octets for the supported cypher suites.

```
$$GP-Claims //= (
    &(gp_se_isd_cci_lfdb_encryption_cypher_suites : 160) => bstr
)
```

6.7.12 SE ISD CCI Token Cypher Suites

The SE ISD CCI Token Cypher Suites Claim defines the supported cypher suites for tokens.

The claim key SHALL be 161 and the claim value SHALL be represented as a CBOR Byte String containing octets for the supported cypher suites.

```
$$GP-Claims //= ( &(gp_se_isd_cci_token_cypher_suites: 161) => bstr )
```

6.7.13 SE ISD CCI Receipt Cypher Suites

The SE ISD CCI Receipt Cypher Suites Claim defines the supported cypher suites for receipts.

The claim key SHALL be 162 and the claim value SHALL be represented as a CBOR Byte String containing octets for the supported cypher suites.

```
$$GP-Claims //= ( &(gp_se_isd_cci_receipt_cypher_suites: 162) => bstr )
```

6.7.14 SE ISD CCI DAP Cypher Suites

The SE ISD CCI DAP Cypher Suites Claim defines the supported cypher suites for Data Authentication Patterns.

The claim key SHALL be 163 and the claim value SHALL be represented as a CBOR Byte String containing octets for the supported cypher suites.

```
$$GP-Claims //= ( &(gp_se_isd_cci_dap_cypher_suites: 163) => bstr )
```

6.7.15 SE ISD CCI Key Parameters

The SE ISD CCI Key Parameters Claim provides a list of key parameter references.

The claim key SHALL be 164 and the claim value SHALL be represented as a CBOR Byte String containing octets for the key parameter references.

```
$$GP-Claims //= ( &(gp_se_isd_cci_key_parameters: 164) => bstr )
```

6.7.16 SE Secure Element Type

The SE Secure Element Type Claim defines the type of the Secure Element.

The claim key SHALL be 165 and the claim value SHALL be represented using the SE Secure Element Type defined in section 6.1.5.

```
$$GP-Claims //= (
    &(gp_se_secure_element_type : 165) => gp_se_secure_element_type_type
)
```

6.7.17 SE Operating System Vendor

The SE Operating System Vendor Claim contains the name of the OS vendor.

The claim key SHALL be 166 and the claim value SHALL be represented as a CBOR Text String containing the octets for the UTF-8 encoded OS vendor name.

```
$$GP-Claims //= ( &(gp_se_operating_system_vendor: 166) => tstr )
```

6.7.18 SE Operating System Version

The SE Operating System Version Claim contains the version of the OS.

The claim key SHALL be 167 and the claim value SHALL be represented as a CBOR Text String containing the octets for the UTF-8 encoded OS version.

```
$$GP-Claims //= ( &(gp_se_operating_system_version: 167) => tstr )
```

6.7.19 SE Chip Manufacturer

The SE Chip Manufacturer Claim contains the name of the chip manufacturer (or in the case of an integrated SE, the name of the SoC manufacturer).

The claim key SHALL be 168 and the claim value SHALL be represented as a CBOR Text String containing the octets for the UTF-8 encoded chip manufacturer name.

```
$$GP-Claims //= ( &(gp_se_chip_manufacturer: 168) => tstr )
```

6.7.20 SE Chip Serial Number

The SE Chip Version Claim contains the serial number of the chip (or in the case of an integrated SE, the serial number of the SoC).

The claim key SHALL be 169 and the claim value SHALL be represented as a CBOR Byte String containing the octets for the serial number.

```
$$GP-Claims //= ( &(gp_se_chip_serial_number: 169) => bstr )
```

6.7.21 SE Chip Version

The SE Chip Version Claim contains the version of the chip (or in the case of an integrated SE, the version of the SoC).

The claim key SHALL be 170 and the claim value SHALL be represented as a CBOR Text String containing the octets for the UTF-8 encoded chip version.

```
$$GP-Claims //= ( &(gp_se_chip_version: 170) => tstr )
```

6.7.22 SE Card Lifecycle State

The SE Card Lifecycle State Claim defines the lifecycle state of the Secure Element.

The claim key SHALL be 171 and the claim value SHALL be encoded as an SE Card Lifecycle State Type as specified in section 6.1.6.

```
$$GP-Claims //= (
    &(gp_se_card_lifecycle_state: 171) => gp_se_card_lifecycle_state_type
)
```

6.7.23 SE Card Content Management Restriction Parameters

The SE Card Content Management Restriction Parameters Claim contains a list of content management restrictions.

The claim key SHALL be 172 and the claim value SHALL be represented as a CBOR Byte String containing the octets of the content management restrictions.

```
$$GP-Claims //= ( &(gp_se_ccm_restriction_parameters: 172) => bstr )
```

6.7.24 SE Supplementary Security Domains

The SE Supplementary Security Domains Claim contains a list of Supplementary Security Domains installed on the Secure Element.

The claim key SHALL be 173 and the claim value SHALL be represented using the SE Supplementary Security Domain Type defined in section 6.1.6.

```
$$GP-Claims //= ( &( gp_se_supplementary_security_domains: 173  ) =>
                    gp_se_supplementary_security_domains_type )
```

6.7.25 SE CASD Certificate Store

The SE CASD Certificate Store Claim contains certificates held by the Controlling Authority Security Domain.

The claim key SHALL be 174 and the claim value SHALL be represented as a CBOR Byte String containing the octets of the CASD certificate store, formatted as defined by the ASN.1 encoding in [GPCS].

```
$$GP-Claims //= ( &(gp_se_casd_certificate_store: 174) => bstr )
```

6.8 Claims Applicable to Trusted Execution Environments

This section defines Claims which correspond to properties of Trusted Execution Environments conforming to the TEE Internal Core API Specification ([TEE Core]) and the TEE Management Framework ([TMF]).

6.8.1 TEE Platform Label

The TEE Platform Label Claim gives an indication about the certification by GlobalPlatform of the TEE ([TMF] `Tee::teePlatformLabel`).

The claim key SHALL be 200 and the claim value SHALL be represented as a CBOR Text String containing the UTF-8 encoded label.

```
$$GP-Claims //= ( &(gp_tee_platform_label: 200) => tstr )
```

6.8.2 TEE Root Security Domains

The TEE Root Security Domains Claim contains a list of root Security Domains (rSDs) installed in the TEE ([TMF] `Tee::roots`).

The claim key SHALL be 201 and the claim value SHALL be represented as a CBOR Array with zero or more entries, each of which being a CBOR Byte String containing the octets that form the UUID of a relevant root Security Domain.

```
$$GP-Claims //= ( &(gp_tee_root_security_domains: 201) => [* bstr] )
```

6.8.3 TEE Lifecycle State

The TEE Lifecycle State Claim gives an indication of the lifecycle state of the TEE ([TMF] `Tee::state`).

The claim key SHALL be 202 and the claim value SHALL be encoded using the TEE Lifecycle State Type defined in section 6.1.7.

```
$$GP-Claims //= (
    &(gp_tee_lifecycle_state: 202) => gp_tee_lifecycle_state_type
)
```

6.8.4 TEE Device Name

The TEE Device Name Claim contains the name of the device ([TMF] `Tee::device::name`).

The claim key SHALL be 203 and the claim value SHALL be a CBOR Text String containing the UTF-8 encoded device name.

```
$$GP-Claims //= ( &(gp_tee_device_name: 203) => tstr )
```

6.8.5 TEE Device Manufacturer

The TEE Device Manufacturer Claim contains the name of the firmware manufacturer ([TMF] `Tee::device::manufacturer`).

The claim key SHALL be 204 and the claim value SHALL be a CBOR Text String containing the UTF-8 encoded device manufacturer name.

```
$$GP-Claims //= ( &(gp_tee_device_manufacturer: 204) => tstr )
```

6.8.6 TEE Device Type

The TEE Device Type Claim contains the type of the device ([TMF] `Tee::device::type`).

The claim key SHALL be 205 and the claim value SHALL be a CBOR Text String containing the UTF-8 encoded device type.

```
$$GP-Claims //= ( &(gp_tee_device_type: 205) => tstr )
```

6.8.7 TEE Trusted OS Name

The TEE Trusted OS Name Claim contains the name of the Trusted OS ([TMF] `Tee::TrustedOS::name`).

The claim key SHALL be 206 and the claim value SHALL be a CBOR Text String containing the UTF-8 encoded Trusted OS name.

```
$$GP-Claims //= ( &(gp_tee_trusted_os_name: 206) => tstr )
```

6.8.8 TEE Trusted OS Architectures

The TEE Trusted OS Architectures Claim includes details of instruction sets and architectures which can be used by Trusted Applications running in the TEE.

The claim key SHALL be 207 and the claim value SHALL be a TEE Trusted OS Architectures Type defined in section 6.1.9.

```
$$GP-Claims //= (
      &(gp_tee_trusted_os_architectures: 207) =>
        gp_tee_trusted_os_architectures_type
)
```

6.8.9 TEE Trusted OS Options

The TEE Trusted OS Options Claim contains a list of options supported by the TEE ([TMF] `Tee::TrustedOS::options`).

The claim key SHALL be 208 and the claim value SHALL be a TEE Options Type, specified in section 6.1.11.

```
$$GP-Claims //= (
    &(gp_tee_trusted_os_options: 208) => gp_tee_options_type
)
```

6.8.10 TEE Optional APIs

The TEE Optional APIs Claim contains a list of optional APIs implemented by the TEE ([TMF] `Tee::optionalApis`).

The claim key SHALL be 209 and the claim value SHALL be a TEE Options Type, specified in section 6.1.11.

```
$$GP-Claims //= (
    &(gp_tee_optional_apis: 209) => gp_tee_options_type
)
```

6.8.11 TEE Implementation Properties

The TEE Implementation Properties Claim contains a list of TEE properties ([TMF] `Tee::teeImplementationProperties`).

The claim key SHALL be 210 and the claim value SHALL be a TEE Implementation Properties Type (section 6.1.12).

```
$$GP-Claims //= (
      &(gp_tee_implementation_properties: 210) =>
        gp_tee_implementation_properties_type
}
```

6.8.12 TEE Trusted Applications

The TEE Trusted Applications Claim contains a list of Trusted Applications installed in the TEE.

The claim key SHALL be 211 and the claim value SHALL be a TEE Trusted Applications Type (section 6.1.15).

```
$$GP-Claims //= (
    &(gp_tee_trusted_applications: 211) =>
      gp_tee_trusted_applications_type
)
```

6.8.13 TEE Security Domains

The TEE Security Domains Claim contains a list of Security Domains present in the TEE.

The claim key SHALL be 212 and the claim value SHALL be a TEE Security Domains Type (section 6.1.17).

```
$$GP-Claims //= (
    &(gp_tee_security_domains: 212) =>
      gp_tee_security_domains_type )
)
```

6.8.14 TEE Client Properties

The TEE Client Properties Claim contains properties related to the TEE which may be of use to clients in the REE, particularly when there is more than one accessible TEE on a system.

The claim key SHALL be 213 and the claim value SHALL be a TEE Client Properties Type (section 6.1.22).

```
$$GP-Claims //= (
    &(gp_tee_client_properties: 213)=>
      gp_tee_client_properties_type )
)
```

7 CLAIMS SETS

A Claims Set is an aggregation of Claims from an entity which can be regarded as a group. If signed or encrypted, all Claims within a Claims Set will be processed with the same algorithm, key material, and context information.

A Claims Set is represented as a CBOR Map, each element of which represents a single Claim. Claims other than those defined in section 6 can be included, but their interpretation is out of scope of this specification.

The value field of most Claims is a straightforward parameter. However, some types of Claim introduce hierarchy, as defined below.

7.1 Subclaims

Some Claims only have a defined meaning when associated with another Claim. These are referred to as Subclaims. Note that a Subclaim has exactly the same structure as a normal Claim. The association is made by including the Subclaims in the Claim Value of a normal Claim. Subclaims SHOULD only be included in the Claim Value of a Claim for which their meaning is defined. If a Claims Set includes Subclaims not included in the Claim Value of a Claim for which their meaning is defined, those Subclaims MAY be ignored.

It is possible to have more than one Claim that has Subclaims in a given Claims Set, but because Claim Keys need to be unique in a CBOR Map, all related Subclaims have to be grouped under a single Claim.

To prepare related Subclaims for use in a Claims Set:

1. Assemble all Subclaims that are related to a single Claim into a nested Claims Set in the normal form of a CBOR Map (even if there is only a single Subclaim).
2. Create a Claim with the relevant Claim Key and the CBOR Map created in step 1 as the Claim Value.

To parse Subclaims included in a Claim present in a Claims Set:

1. If the Claim Key indicates that it can include Subclaims, extract the Subclaims by interpreting the Claim Value as a CBOR Map.
2. Treating the CBOR Map as a nested Claims Set, parse each Subclaim in the context of the Claim Key obtained in step 1.

7.2 Submodules

Where a device has more than one platform capable of generating evidence for inclusion in an Entity Attestation Token, it is possible to group claims sets or tokens from the various sources together using the submods Claim. The Claims Sets or Tokens from each source are grouped together as an array, then the array is included in the Claim Value of a submods Claim.

Because Claim Keys need to be unique in a CBOR Map, only a single submods Claim can be present, so Claims Sets and Tokens from all submodules have to be grouped under a single submods Claim.

Claims Sets, Entity Attestation Tokens, and Detached Submodule Digests can be included in the submods Claim, and effectively a Claims Set is included as an untagged Unendorsed Claims Set. An untagged Unendorsed Claims Set takes the form of a CBOR Map, and an untagged Entity Attestation Token or Detached Submodule Digest takes the form of a CBOR Array, so it may be distinguished at the CBOR level. Although not described in [draft-ietf-rats-eat], it is also possible to assist the parser by including the Unendorsed Claims Set tag and the CWT tag plus the COSE tag for an Entity Attestation Token.

To prepare a Claims Set from a submodule for inclusion in a submods Claim:

1. Create a Submod Name Claim with a descriptive label as the Claim Value.
2. Optionally create a Security Rating Claim with values suitable for the submodule, with the restriction that the submodule cannot be marked as higher security than that of the EAS.
3. Optionally create other Claims with additional information relevant to the submodule.
4. Assemble the Claims from steps 1, 2, and 3 together with all the Claims from the submodule into an Unendorsed Claims Set in the normal form of a CBOR Map (even if there is only a single Claim).
5. Optionally prefix the Claims Set with the Unendorsed Claims Set tag.

To prepare an Entity Attestation Token from a submodule for inclusion in a submods Claim:

1. If not already present, optionally prefix the token with the CWT tag and the appropriate COSE tag.

To prepare a Detached Submodule Digest from a submodule for inclusion in a submods Claim:

1. No further action is required.

To prepare Claims Sets, Entity Attestation Tokens, and Detached Submodule Digests from all submodules for use in a Claims Set:

1. For each Claims Set, Entity Attestation Token, or Detached Submodule Digest, prepare a nested Entity Attestation Parcel as described above.
2. Assemble all the nested Entity Attestation Parcels from step 1 into a CBOR Array (even if there is only a single Entity Attestation Parcel).
3. Create a submods Claim with the CBOR Array created in step 2 as the Claim Value.

To parse submodule Claims Sets included in a submods Claim present in a Claims Set:

1. Extract the Claims Sets for all submodules by interpreting the Claim Value as a CBOR Array.
2. For each element in the CBOR Array, extract an Entity Attestation Parcel by interpreting the array entry as an Unendorsed Claims Set, Entity Attestation Token, or Detached Submodule Digest.

To parse a Claims Set from a submodule included in a nested Unendorsed Claims Set:

1. If present, check that the tag value matches the Unendorsed Claims Set tag value, and exit with an error if it does not.
2. Check that the CBOR Major Type is Map, and exit with an error if it is not.
3. Treat the CBOR Map as a nested Claims Set.
4. If the nested Claims Set does not contain a Submod Name Claim, the Claims from the submodule MAY be ignored.

To parse an Entity Attestation Token from a submodule included in a nested Entity Attestation Token:

1. If present, check that the tag value matches the Entity Attestation Token tag, and exit with an error if it does not.
2. If an Entity Attestation Token is present, check that it is followed by a valid COSE tag, and exit with an error if not.
3. Check that the CBOR Major Type is Array, and exit with an error if it is not.
4. Treat the CBOR Array as an Entity Attestation Token.

To parse a Detached Submodule Digest from a submodule included in a nested Detached Submodule Digest:

1. No further action is required.

8 UNENDORSED CLAIMS SETS

An Unendorsed Claims Set is based on the Unprotected CWT Claims Set [draft-birkholz-rats-uccs] which is not based on COSE, but does use CBOR for encoding. It includes the following elements:

- The tag used to identify the content as a Claims Set (required only if the meaning of the payload is not otherwise obvious from context)
- An unencapsulated Claims Set (the payload)

8.1 Construction

The Unendorsed Claims Set is constructed as follows:

1. Create the unencapsulated Claims Set:
   a. Create the Claims Set as a CBOR Map.
2. Optionally prefix the CBOR Map constructed in step 1 with the UCCS tag.

8.2 Interpretation

To obtain a Claims Set from the content of an Unendorsed Claims Set, the following steps are taken:

1. Interpret the string of bytes according to [RFC 8949], and exit with an error if it is not well formed.
2. If a tag is present, check that it matches the UCCS value, and exit with an error if it does not.
3. Check that the CBOR Major Type is Map, and exit with an error if it is not.
4. Interpret the CBOR Map as a Claims Set.

9 ENTITY ATTESTATION TOKENS

An Entity Attestation Token is based on the CBOR Web Token (CWT) [RFC 8152] which itself uses the COSE Messages defined in [RFC 8152], and as such includes at least the following elements:

- A set of protected header parameters
- A set of unprotected header parameters
- The content of the message

The format of each element is defined in [RFC 8152].

An EAT can also include additional elements, as defined in [RFC 8152], for each of the defined COSE Messages.

The unprotected header can contain countersignatures.

The content of the message is always a CBOR Byte String which encapsulates the CBOR encoding of the CBOR Map that contains the Claims.

An Entity Attestation Token is endorsed in one of the following ways.

- A Signed Entity Attestation Token consists of the encapsulated Claims Set as the content plus at least one signature; it is defined in section 10.
- An Encrypted Entity Attestation Token consists of the ciphertext equivalent of the Claims Set as the content, and may contain key distribution information; it is defined in section 11.
- A MACed Entity Attestation Token consists of the encapsulated Claims Set as the content plus a MAC, and may contain key distribution information; it is defined in section 12.

Section 13 defines how to apply more than one type of endorsement.

10 SIGNED ENTITY ATTESTATION TOKENS

This specification defines how to create a signed Token but imposes no requirements on which algorithms need to be supported in any given system. Configuration documents could mandate specific signing algorithms for particular use cases.

In the GlobalPlatform usage, Entity Attestation Tokens SHALL be signed by a Root of Trust.

10.1 Prerequisites

When using a PKI scheme, the recipient will only be able to verify a signature if it has access to the public key corresponding to the private key used for signing. There are many well-established ways by which public key material can be distributed, and this specification takes as a prerequisite that this has been done.

10.2 Context

The following context is required in order to construct a Signed Entity Attestation Token.

- Number of signatures required
- Algorithm for each signature
- Key identifier for each signature
- Any other body or recipient header parameters
- Any external data

The choice of algorithm for each signature is dependent on security requirements and the capability of the recipient, and is out of scope of this document.

The inclusion of header parameters not directly related to the signing process, and any policies regarding verifying or using any such parameters, are application dependent, and are out of scope of this document.

10.3 Construction

10.3.1 Creating the Content

The content is created as follows:

1. Create the CBOR encoding for the Claims Set as a series of bytes.
2. Create a CBOR Byte String with the payload set to the series of bytes generated in step 1.

10.3.2 Signing the EAT

For each required signer, a signature is created following the process defined in [RFC 8152]. At a minimum, header parameters are created which contain the algorithm and key identifier used in generating the signature. Other header parameters can also be included.

If only a single signature is required, then a `COSE_Sign1` message SHOULD be used because the size of the CBOR representation is slightly smaller. In all other cases, a `COSE_Sign` message SHALL be used.

In the case of `COSE_Sign1`, the Signed EAT is constructed as follows:

1. Construct protected and unprotected headers from parameters including the signature algorithm.
2. Add the content constructed in section 10.3.1.
3. Create the signature following the process defined in [RFC 8152].
4. Add the signature as a fourth element in the `COSE_Sign1` Message.

In the case of `COSE_Sign`, the Signed EAT is constructed as follows:

1. For each signer:
   a. Construct protected and unprotected headers from parameters including the signature algorithm and the key identifier.
   b. Create the signature following the process defined in [RFC 8152].
   c. Create a `COSE_Signature` from the protected and unprotected headers plus the signature itself as the content.
2. Create an array including each `COSE_Signature` created in step 1.
3. Construct body protected and unprotected headers which can contain header parameters unrelated to the signatures.
4. Add the content constructed in section 10.3.1.
5. Add the array created in step 2 as a fourth element of the `COSE_Sign` Message.

In both cases, the COSE Message can be tagged with the appropriate value if the nature of the Signed EAT is not obvious from context, as defined in section 14.1.

10.4 Interpretation

If the nature of the Signed EAT is known from context, for example if it comes from a COSE tagged EAT as defined in section 14.2, this can be used to check the structure of the COSE Message in the content data of the CBOR Tag. Otherwise, the determination is made by checking the fourth item.

- If this is a CBOR Array of `COSE_Signature`, handle the EAT as if tagged with the `COSE_Sign` value.
- If this is a CBOR Byte String, handle the EAT as if tagged with the `COSE_Sign1` value.
- Otherwise exit with the code “UNVERIFIED”.

If the Signed EAT contains a `COSE_Sign1` message, the verification parameters, including the algorithm and key identifier to be used, are determined from the body protected and unprotected headers, and the signature is obtained from the fourth data element of the `COSE_Sign1` Message.

If the Signed EAT contains a `COSE_Sign` message, then for each signature, the necessary information is determined from the protected and unprotected headers plus the content element of each `COSE_Signature` in the array.

In both cases, the payload is in the body content field.

The following applies for each signature in the Signed EAT:

- Check whether the algorithm used for the signature verification is supported, and exit with the code “UNVERIFIED” if it is not.
- Check whether the key used for the signature verification is available, and exit with the code “UNVERIFIED” if it is not.
- Verify the signature following the process defined in [RFC 8152], and exit with the code “INVALID” if the verification is unsuccessful.
- Exit with the code “VALID”.

This specification imposes no requirements on:

- How to proceed if a signature cannot be verified because a key or algorithm is not available
- Whether all signatures need to be verified or only a subset
- How to process any header parameters unrelated to the signature verification

Configuration documents may choose to impose requirements in this area.

Once verification of the Signed EAT is complete, the content is interpreted as follows:

1. Check that the CBOR Major Type is Byte String, and exit with an error if it is not.
2. Extract the payload of the byte string as a series of bytes.
3. Interpret the string of bytes according to [RFC 8949], and exit with an error if it is not well formed.
4. Check that the CBOR Major Type is Map, and exit with an error if it is not.
5. Interpret the CBOR Map as a Claims Set.

11 Encrypted Entity Attestation Tokens

This specification defines how to create an encrypted Token but imposes no requirements on which algorithms need to be supported in any given system. Configuration documents could mandate specific encryption algorithms for particular use cases.

In the GlobalPlatform usage, Entity Attestation Tokens SHALL be encrypted and decrypted by a Root of Trust.

11.1 Prerequisites

In order to be able to decrypt an Encrypted Entity Attestation Token, the recipient needs either to possess a suitable key or to be given the information necessary to construct one.

11.1.1 Symmetric Encryption

The simplest case of symmetric encryption is where both parties possess a common shared secret by some means out of scope of this specification. Such means include provisioning during manufacturing, a previous key agreement procedure, or a previous key distribution procedure.

Alternatively, each party could possess a static public key generated by the other. In this case, a common shared secret can be constructed using key agreement providing that information about the keys used is included in the Encrypted EAT.

Finally, the generator of the EAT could possess a static public key for the recipient, but either knows that the recipient does not possess a public key from the generator, or chooses to use an ephemeral key. Here, a common shared secret can be constructed using key agreement providing that information about the recipient key and the value of the ephemeral public key are included in the Encrypted EAT.

In all cases, including that of the fixed shared secret, key derivation can be applied for a specific use, providing that context information is already known by both parties, or included in the headers of the Encrypted EAT.

11.1.2 Asymmetric Encryption

The simplest case of asymmetric encryption is where the generator of the EAT possesses a public key for a private key held by the recipient by some means out of scope of this specification. In this case, the EAT can contain the name of the private key, or this can be omitted if the key to use is implicitly known.

Alternatively, the generator can use key wrapping to send the private key to the recipient. The recipient can unwrap the private key using a shared secret which is either pre-shared or derived from header information in the Encrypted EAT.

11.2 Context

The following context is required in order to construct an Encrypted Entity Attestation Token.

- Encryption algorithm
- Optionally, key distribution algorithm
- Key identifier(s)
- Any other body or recipient header parameters
- Any external data

The choice of encryption algorithm is dependent on security requirements and the capability of the recipient, and is out of scope of this document. The choice of key distribution algorithm is dependent on security requirements and key availability, and is out of scope of this document.

The inclusion of header parameters not directly related to the encryption process, and any policies regarding verifying or using any such parameters, are application dependent, and are out of scope of this document.

11.3 Construction

11.3.1 Creating the Content

The content is created as follows:

1. Create the CBOR encoding for the Claims Set as a series of bytes.
2. Create a CBOR Byte String with the payload set to the series of bytes generated in step 1.

11.3.2 Encrypting the EAT

Ciphertext is created following the process defined in [RFC 8152]. At a minimum, header parameters are created which contain the algorithm and key identifier(s) or key value used in generating the ciphertext. Certain algorithms can require the inclusion of additional header parameters such as an initial value. Other header parameters can also be included.

If a shared secret is already known to both the Relying Party and the Entity Attestation Service, and no key derivation is required, then a `COSE_Encrypt0` message SHOULD be used because the size of the CBOR representation is slightly smaller. In all other cases, a `COSE_Encrypt` message SHALL be used.

In the case of `COSE_Encrypt0`, the Encrypted EAT is constructed as follows:

1. Construct protected and unprotected headers from parameters including the encryption algorithm and any additional values required by the algorithm.
2. Create the ciphertext from the content constructed in section 11.3.1 plus any associated data, following the process defined in [RFC 8152].
3. Add the ciphertext as the content of the `COSE_Encrypt0` Message.

In the case of `COSE_Encrypt`, the Encrypted EAT is constructed as follows:

1. Construct recipient protected and unprotected headers from parameters including the key distribution algorithm, the key identifiers or values, and any additional values such as salt or context information.
2. Create a `COSE_Recipient` from the protected and unprotected headers plus the ciphertext as the content.
3. Create an array of `COSE_Recipients` with the single entry created in step 2.
4. Construct body protected and unprotected headers from parameters including the encryption algorithm and any additional values required by the algorithm.
5. Create the ciphertext from the content constructed in section 11.3.1 plus any associated data, using a key generated using a mechanism defined in one of the recipients, following the process defined in [RFC 8152].
6. Add the array of `COSE_Recipients` constructed in step 3 as a fourth element of the `COSE_Encrypt` Message.

In both cases, the COSE Message can be tagged with the appropriate value if the nature of the Encrypted EAT is not obvious from context, as defined in section 14.1.

11.4 Interpretation

If the nature of the Encrypted EAT is known from context, for example if it comes from a COSE tagged EAT as defined in section 14.2, then the nature can be used to check the structure of the COSE Message in the content data of the CBOR Tag. Otherwise, the determination is made by checking the fourth item.

- If there is no fourth item present, handle the EAT as if tagged with the `COSE_Encrypt0` value.
- If the fourth item is a CBOR Array of `COSE_Recipients`, handle the EAT as if tagged with the `COSE_Encrypt` value.
- Otherwise, either handle the EAT as if tagged with the `COSE_Encrypt0` value (ignoring the information in the fourth item), or exit with the code “UNVERIFIED”.

The decryption parameters including the algorithm to be used are determined from the body protected and unprotected headers. The ciphertext is obtained from the body content. If the Encrypted EAT contains a `COSE_Encrypt` Message, the key distribution parameters including the algorithm to be used are determined from the protected and unprotected headers of the `COSE_Recipients`. Otherwise, the key material to use is expected to be present and known from context.

If the Encrypted EAT contains a `COSE_Encrypt` message, the decryption key is constructed as follows:

1. Check whether the algorithm used for key distribution is supported, and exit if it is not.
2. Check whether all indicated keys are available, and exit if they are not.
3. Construct the key using the algorithm, keys, and any other relevant parameters in the recipient protected and unprotected headers.

If there is more than one recipient present, the verifier may choose any one to derive the decryption key based on which mechanisms it supports.

The ciphertext is verified as follows.

1. Check whether the algorithm used for decryption is supported, and exit with the code “UNVERIFIED” if it is not.
2. Check whether the key used for decryption is available, and exit with the code “UNVERIFIED” if it is not.
3. Decrypt the ciphertext following the process defined in [RFC 8152], and exit with the code “INVALID” if the verification is unsuccessful.
4. Exit with the plaintext and the code “VALID”.

This specification imposes no requirements on:

- How to process any header parameters unrelated to key distribution or decryption

Configuration documents may choose to impose requirements in this area.

Once verification of the Encrypted EAT is complete, the plaintext is interpreted as follows:

1. Check that the CBOR Major Type is Byte String, and exit with an error if it is not.
2. Extract the payload of the byte string as a series of bytes.
3. Interpret the string of bytes according to [RFC 8949], and exit with an error if it is not well formed.
4. Check that the CBOR Major Type is Map, and exit with an error if it is not.
5. Interpret the CBOR Map as a Claims Set.

12 MACED ENTITY ATTESTATION TOKENS

This specification defines how to create a MACed Token but imposes no requirements on which algorithms need to be supported in any given system. Configuration documents could mandate specific MAC algorithms for particular use cases.

12.1 Prerequisites

In order to be able to authenticate the MAC included in an Encrypted Entity Attestation Token, the recipient needs either to possess a suitable key, or to be given the information necessary to construct one. The basis for MAC authentication key generation is identical to that for decryption key generation in section 11.1.

12.2 Context

The following context is required in order to construct a MACed Entity Attestation Token.

- MAC algorithm
- Optionally, key distribution algorithm
- Key identifier(s)
- Any other body or recipient header parameters
- Any external data

The choice of MAC algorithm is dependent on security requirements and the capability of the recipient, and is out of scope of this document. The choice of key distribution algorithm is dependent on security requirements and key availability, and is out of scope of this document.

The inclusion of header parameters not directly related to the MAC process, and any policies regarding verifying or using any such parameters, are application dependent, and are out of scope of this document.

12.3 Construction

12.3.1 Creating the Content

The content is created as follows:

1. Create the CBOR encoding for the Claims Set as a series of bytes.
2. Create a CBOR Byte String with the payload set to the series of bytes generated in step 1.

12.3.2 MACing the EAT

A Message Authentication Code (MAC) is created following the process defined in [RFC 8152]. At a minimum, header parameters are created which contain the algorithm and key identifier(s) or key value used in generating the MAC. Certain algorithms can require the inclusion of additional header parameters. Other header parameters can also be included.

If a shared secret is already known to both the Relying Party and the Entity Attestation Service, and no key derivation is required, then a `COSE_Mac0` message SHOULD be used because the size of the CBOR representation is slightly smaller. In all other cases, a `COSE_Mac` message SHALL be used.

In the case of `COSE_Mac0`, the MACed EAT is constructed as follows:

1. Construct protected and unprotected headers from parameters including the MAC algorithm and any additional values required by the algorithm.
2. Create the MAC from the content constructed in section 11.3.1 plus any associated data, following the process defined in [RFC 8152].
3. Add the MAC as a fourth element in the `COSE_Mac0` Message.

In the case of `COSE_Mac`, the MACed EAT is constructed as follows:

1. Construct recipient protected and unprotected headers from parameters including the key distribution algorithm, the key identifiers or values, and any additional values such as context information.
2. Create a `COSE_Recipient` from the protected and unprotected headers plus the ciphertext as the content.
3. Create an array of `COSE_Recipients` with the single entry created in step 2.
4. Construct body protected and unprotected headers from parameters including the encryption algorithm and any additional values required by the algorithm.
5. Create the MAC from the content constructed in section 12.3.1 plus any associated data, using a key generated using a mechanism defined in one of the recipients, following the process defined in [RFC 8152].
6. Add the MAC created in step 5 as a fourth element of the `COSE_Mac` message.
7. Add the array of `COSE_Recipients` constructed in step 3 as a fifth element of the `COSE_Mac` message.

In both cases, the COSE Message can be tagged with the appropriate value if the nature of the MACed EAT is not obvious from context, as defined in section 14.1.

12.4 Interpretation

If the nature of the MACed EAT is known from context, for example if it comes from a COSE tagged EAT as defined in section 14.2, this can be used to check the structure of the COSE Message in the content data of the CBOR Tag. Otherwise, the determination is made by checking the fifth items.

- If there is no fifth item present, handle the EAT as if tagged with the `COSE_Mac0` value.
- If the fifth item is a CBOR Array of `COSE_Recipients`, handle the EAT as if tagged with the `COSE_Mac` value.
- Otherwise, either handle the EAT as if tagged with the `COSE_Mac0` value (ignoring the information in the fourth item), or exit with the code “UNVERIFIED”.

The MAC authentication parameters including the algorithm to be used are determined from the body protected and unprotected headers. The Claims Set is obtained from the body content. If the MACed EAT contains a `COSE_Mac` message, the key distribution parameters including the algorithm to be used are determined from the protected and unprotected headers of the `COSE_Recipients`. Otherwise, the key material to use is expected to be present and known from context.

If the MACed EAT contains a `COSE_Mac` message, the MAC authentication key is constructed as follows:

1. Check whether the algorithm used for key distribution is supported, and exit if it is not.
2. Check whether all indicated keys are available, and exit if they are not.
3. Construct the key using the algorithm, keys, and any other relevant parameters in the recipient protected and unprotected headers.

If there is more than one recipient present, the verifier may choose any one to derive the MAC authentication key based on which mechanisms it supports.

The MAC is authenticated as follows.

1. Check whether the algorithm used for authentication is supported, and exit with the code “UNVERIFIED” if it is not.
2. Check whether the key used for authentication is available, and exit with the code “UNVERIFIED” if it is not.
3. Authenticate the MAC following the process defined in [RFC 8152], and exit with the code “INVALID” if the authentication is unsuccessful.
4. Exit with the code “VALID”.

This specification imposes no requirements on:

- How to process any header parameters unrelated to key distribution or MAC authentication

Configuration documents may choose to impose requirements in this area.

Once authentication of the MACed EAT is complete, the content is interpreted as follows:

1. Check that the CBOR Major Type is Byte String, and exit with an error if it is not.
2. Extract the payload of the byte string as a series of bytes.
3. Interpret the string of bytes according to [RFC 8949], and exit with an error if it is not well formed.
4. Check that the CBOR Major Type is Map, and exit with an error if it is not.
5. Interpret the CBOR Map as a Claims Set.

13 MULTIPLY ENDORSED ENTITY ATTESTATION TOKENS

In certain circumstances, it might be necessary to apply two types of endorsement to an Entity Attestation Token. For example, a token might need to be signed for authenticity reasons, then encrypted for confidentiality reasons.

A multiply endorsed Entity Attestation Token is constructed as follows:

1. An Entity Attestation Token is constructed using one of the methods defined in section 9.
2. A Claims Set is created that contains a "`submods`" claim containing the EAT constructed in step 1.
3. An Entity Attestation Token containing the Claims Set created in step 2, using one of the methods defined in section 9.

This approach can continue with third or higher numbers of endorsements.

A multiply endorsed Entity Attestation Token is parsed as follows:

1. The Entity Attestation Token is parsed using the appropriate method defined above.
2. The content of the "`submods`" Claim is extracted and interpreted as an Entity Attestation Token.
3. The nested Entity Attestation Token is parsed using the appropriate method defined above.

14 TAGGING ENTITY ATTESTATION TOKENS

Unless context makes it certain that an Entity Attestation Token will be recognized for what it is, a defined tag SHOULD be used to assist in parsing.

Any untagged Entity Attestation Token MAY be tagged.

An Entity Attestation Token that has been created using the process described in the sections above has no tags, and contains a `COSE_Untagged_Message` (specifically, a `COSE_Sign`, a `COSE_Sign1`, a `COSE_Encrypt`, a `COSE_Encrypt0`, a `COSE_Mac`, or a `COSE_Mac0` Message). If a COSE tag is added, the EAT contains a `COSE_Tagged_Message` (specifically a `COSE_Sign_Tagged`, a `COSE_Sign1_Tagged`, a `COSE_Encrypt_Tagged`, a `COSE_Encrypt0_Tagged`, a `COSE_Mac_Tagged`, or a `COSE_Mac0_Tagged` Message).

The Entity Attestation Token specification [draft-ietf-rats-eat] defines an EAT as being a CBOR Web Token [RFC 8392]. This specification follows [draft-ietf-rats-eat] in including the value 61 as the tag value. As required by [RFC 8392], if the Entity Attestation Token is prefixed by the CWT tag, the CWT tag SHALL be immediately followed by a COSE tag.

14.1 Construction

A Tagged Entity Attestation Token is constructed as follows:

1. Create an untagged EAT as defined above.
2. Create a COSE tagged EAT by prefixing the untagged EAT built in step 1 with the appropriate COSE tag as defined in [RFC 8949].
3. If the CWT tag is to be included, create the tagged EAT by prefixing the COSE tagged EAT built in step 2 with the CWT tag.
4. Otherwise, the tagged EAT is just the COSE tagged EAT built in step 2.

14.2 Interpretation

A Tagged Entity Attestation Token is interpreted as follows:

1. If the tag value of the tagged EAT matches the CWT tag, the COSE tagged EAT is the nested CBOR Data Item as defined in [RFC 8949], otherwise the COSE tagged EAT is just the tagged EAT.
2. Check that the CBOR Major Type of the COSE tagged EAT built in step 1 is Tag, and that the tag value matches a defined COSE value, and exit with an error if not.
3. The untagged EAT is the nested CBOR Data Item of the COSE tagged EAT build in step 2.
4. Interpret the untagged EAT as defined above.

15 DETACHED EAT BUNDLES

Detached EAT Bundles are defined in [draft-ietf-rats-eat]. These are conceptually similar to Entity Attestation Tokens, but the use is different. An example where a Detached EAT Bundle might be used is when attesting a large amount of data. The Entity Attestation Token returned by the Attestation Service can include a much smaller amount of information in the form of a Detached Submodule Digest inside a Submods claim. This would typically provide a signed hash of the large amount of data. A Relying Party can then be sent the actual data by other methods, including the use of a Detached EAT Bundle, although other means can be used.

This means that an Attestation Service can be implemented on a memory-constrained device because it no longer needs to assemble a complete Entity Attestation Token containing the large amount of data. It merely needs enough resources to be able to compute and store the hash value.

The Relying Party can still assess the trustworthiness of the data because it can compute the hash of the data it receives separately, then compare it with the value in the Detached Submodule Digest included in the endorsed Entity Attestation Token. If the data is tampered with in transit, the hash values will not match.

16 DEPENDENCIES ON OTHER SPECIFICATIONS

This specification has specific dependencies on a number of other specifications.

16.1 Dependency on [RFC 8949]

The types and encoding defined in [RFC 8949] are used without modification, although this specification does not reference JSON, JOSE, or JWT. This specification does not require the use of canonical CBOR, although it is possible that some applications may impose this requirement.

16.2 Dependency on [RFC 8152]

Structures defined in [RFC 8152] are used without modification, although this specification does not use those that relate to Message Authentication Codes (MACs).

16.3 Dependency on [RFC 8230]

Parameters defined in [RFC 8230] are used without modification.

16.4 Dependency on [RFC 8392]

The format defined in [RFC 8392] is used without modification. The Claims included from [RFC 8392] are listed in section 6.3.

16.5 Dependency on [draft-ietf-rats-eat]

The format defined in [draft-ietf-rats-eat] is used with the following modifications:

- This specification does not reference JSON, JOSE, or JWT.

The rules for CBOR interoperability defined in [draft-ietf-rats-eat] are used without modification. The Claims included from [draft-ietf-rats-eat] are listed in section 6.3.

16.6 Dependency on [draft-birkholz-rats-uccs]

The tag value defined in [draft-birkholz-rats-uccs] is used without modification.

17 IANA CONSIDERATIONS

17.1 Reuse of Concise Binary Object Representation (CBOR) Registries

The following IANA registries referenced by [RFC 8949] are reused:

- CBOR Simple Values Registry
- CBOR Tags Registry

17.2 Reuse of CBOR Object Signing and Encryption (COSE) Registries

The following IANA registries referenced by [RFC 8152] are reused:

- Tag assignments added to the CBOR Tags Registry
- COSE Header Parameters Registry
- COSE Header Algorithm Parameters Registry
- COSE Algorithms Registry
- COSE Key Common Parameters Registry
- COSE Key Type Parameters Registry
- COSE Key Types Registry
- COSE Elliptic Curves Registry

17.3 Reuse of RSA Algorithms with COSE Messages Registries

The following IANA registries referenced by [RFC 8230] are reused:

- Algorithm assignments added to the COSE Algorithms Registry
- Key Type assignments added to the COSE Key Types Registry
- Key Type Parameter assignments added to the COSE Key Type Parameters Registry

17.4 Reuse of CBOR Web Token (CWT) Registries

The following IANA registries referenced by [RFC 8392] are reused:

- CBOR Web Token (CWT) Claims Registry
- Tag assignments added to the CBOR Tags Registry

17.5 Reuse of Entity Attestation Token (EAT) Registries

The following IANA registries referenced by [draft-ietf-rats-eat] are reused:

- Claim assignments added to the CWT Claims Registry

17.6 Reuse of Unprotected CWT Claims Set (UCCS) Registries

The following IANA registries referenced by [draft-birkholz-rats-uccs] are reused:

- Values for Tags

17.7 Claims Registered by This Document

This document registers a single claim in the IANA CWT registry:

Claim Name: `globalplatform_component`
Claim Description: This claim holds an array of CBOR maps in which each array entry holds a map containing claims about a GlobalPlatform component that is within the boundary of the enclosing Entity Attestation Token.
Claim Key: TBC
Change Controller: IESG
Specification Document: This Document <ADD CROSS REFERENCE>

Annex A CONFIGURATIONS

This document defines Claims and mechanisms for signing and encryption, but imposes no requirements on support. Configuration documents could be created for specific Attestation use cases that do impose restrictions. Such configurations could define:

- Which Claims are mandatory, optional, or not allowed
- Which signing algorithm (if any) is to be used
- Which encryption algorithm (if any) is to be used
- Which MAC algorithm (if any) is to be used
- Which Operations are allowed

For example, a configuration for a TEE Entity Attestation Parcel could include the following:

Table A-1 places restrictions on Claims for a TEE EAP.

Table A-1: Restrictions on Claims (Example)

Claim	Inclusion
TEE API Version	Mandatory
TEE Description	Optional
TEE Device ID	Mandatory

Table A-2 places restrictions on signing algorithms for a TEE EAP. The choice of algorithm is implementation dependent. A TEE Claims Set SHALL be signed, and the algorithm SHALL be indicated in the COSE protected header.

Table A-2: Restrictions on Signing Algorithms (Example)

Signing Algorithm	Use
ECDSA with SHA-256	Optional
ECDSA with SHA-384	Optional
ECDSA with SHA-512	Optional

Table A-3 places restrictions on encryption algorithms for a TEE EAP. The choice of algorithm is implementation dependent. A TEE Claims Set may be encrypted; if it is, the algorithm SHALL be indicated in the COSE protected header.

Table A-3: Restrictions on Encryption Algorithms (Example)

Encryption Algorithm	Use
AES-GCM with at least 256-bit key	Optional
AES-CCM with at least 256-bit key	Optional

Annex B EXAMPLES

This annex provides examples of:

- Claims Sets
- Unendorsed Claims Sets
- Entity Attestation Tokens

B.1 Example Claims Sets

In the examples below, all signatures are computed using the NIST P-256 elliptic curve key:

```
{
1 : 2,
2 : "signatureKey",
-1 : 1,
-2 : h'106221229e9205c5387e3be4ab2045a3921e7531bb59055086516d964236aac8',
-3 : h'6698ef082482665dff8b10825397229093e3dd9e70f0f7b290f40eb2ed672193',
-4 : h'bf14d67e2ddc8e6683ef574961ff698f61cdd11e9d9c167272e61df0844f4a77'
}
```

In the examples below, all ciphertext is computed using the 128-bit symmetric key:

```
{
1 : 4,
2 : "encryptionKey",
-1 : h'02d7e8392c53cbc9121e33749e0cf4d5'
}
```

In the examples below, all MACs are computed using the 256-bit symmetric key:

```
{
1 : 4,
2 : "macKey",
-1 : h' 2923be84e16cd6ae529049f1f1bbe9ebb3a6db3c870c3e99245e0d1c06b747de'
}
```

B.1.1 Example Flat SE Claims Set

```
{
  / Security Rating / 1000 : 15,
  / SE IIN / 1100 : h'11ee11ee11ee11ee11ee11ee11ee11ee',
  / SE CIN / 1101 : h'22dd22dd22dd22dd22dd22dd22dd22dd',
  / SE Secure Channel Protocols / 1106 :
  [
    {
      / SCP Type / 1: 112,
      / SCP Options / 2: h'53ae32'
    },
    {
      / SCP Type / 1: 3,
      / SCP Options / 2: h'91b8a402',
      / SCP SCP03 Keys / 3: h'737337177172'
    },
    {
      / SCP Type / 1: 129,
      / SCP Options / 2: h'a07eba9e8ae3',
```

```
      / SCP SCP81 TLS Suites / 4: h'91b8a402',
      / SCP SCP81 Max Key Length / 5: 128
    }
  ],
  / SE Type / 1115 : 10,
  / SE Card Lifecycle State / 1121 : 10,
  / SE Supplementary Security Domains / 1123 :
  [
    {
      / Application ID / 1: h'00112233445566778899aabbccddeeff',
      / Recognition Data / 2: h'21e4918cda74da12',
      / Key Derivation Data / 3: h'a07eba9e8ae3',
      / Key Information Data / 4: h'737337177172',
      / Default KVN / 5: 3,
      / Default KVN Sequence / 6: h'91b8a402'
    },
    {
      / Application ID / 1: h'112233445566778899aabbccddeeff00',
      / Recognition Data / 2: h'737337177172',
      / Key Derivation Data / 3: h'a9301412789124',
      / Key Information Data / 4: h'02d41e4120db94e1cc24',
      / Default KVN / 5: 18,
      / Default KVN Sequence / 6: h'bd9124afe881a991'
    }
  ]
}
```

CBOR encoded value:

```
a71903e80f19044c5011ee11ee11ee11ee11ee11ee11ee11ee19044d5022dd22dd22dd22dd22d
d22dd22dd22dd19045283a2011870024353ae32a30103024491b8a4020346737337177172a401
18810246a07eba9e8ae3044491b8a40205188019045b0a1904610a19046382a60150001122334
45566778899aabbccddeeff024821e4918cda74da120346a07eba9e8ae3044673733717717205
03064491b8a402a6015011223344556678899aabbccddeeff000246737337171771720347a9301
412789124044a02d41e4120db94e1cc2405120648bd9124afe881a991
```

B.1.2 Example Flat TEE Claims Set

```
{
  / Security Rating / 1000 : 10,
  / TEE Root Security Domains / 1203 :
  [
    h'03a29483e29814de9b214b367af4228b'
  ],
  / TEE Lifecycle State / 1204 : 1,
  / TEE Implementation Properties / 1216 :
  {
    "CryptoLibraryVersion": 293,
    "CryptoLibraryName": "AceCryptoLibrary",
    "AliceLoginCredentials":
    {
      "loginMethod": 7,
      "uuid": h'00ff00ff00ff00ff00ff00ff00ff00ff'
    }
  },
  / TEE Trusted Applications / 1217 :
  [
    {
      / TA Identifier / 1: h'44bb44bb44bb44bb44bb44bb44bb44bb',
```

```
      / TA Parent / 2: h'bb44bb44bb44bb44bb44bb44bb44bb44',
      / TA Lifecycle State / 3: 2,
      / TA Version / 4: "Beta 0.9"
    },
    {
      / TA Identifier / 1: h'33cc33cc33cc33cc33cc33cc33cc33cc',
      / TA Parent / 2: h'cc33cc33cc33cc33cc33cc33cc33cc33',
      / TA Lifecycle State / 3: 0,
      / TA Version / 4: "Production 2.5.1"
    }
  ],
  / TEE Security Domains / 1218 :
  [
    {
      / SD Identifier / 1: h'00112233445566778899aabbccddeeff',
      / SD Recognition Data / 2: h'21e4918cda74da12',
      / SD Key Derivation Data / 3: h'a07eba9e8ae3',
      / SD Key Information Data / 4: h'737337177172',
      / SD Default KVN / 5: 3,
      / SD Default KVN Sequence / 6: h'91b8a402'
    },
    {
      / SD Identifier / 1: h'11223344556778899aabbccddeeff00',
      / SD Recognition Data / 2: h'737337177172',
      / SD Key Derivation Data / 3: h'a9301412789124',
      / SD Key Information Data / 4: h'02d41e4120db94e1cc24',
      / SD Default KVN / 5: 18,
      / SD Default KVN Sequence / 6: h'bd9124afe881a991'
    }
  ]
}
```

CBOR encoded value:

```
a61903e80a1904b3815003a29483e29814de9b214b367af4228b1904b4011904c0a3744372797
0746f4c6962726172795665727369 6f6e1901257143727970746f4c6962726172794e616d6570
41636543727970746f4c6962726172797541 6c6963654c6f67696e43726564656e7469616c73a
26b6c6f67696e4d6574686f6407647575696450 00ff00ff00ff00ff00ff00ff00ff1904c1
82a4015044bb44bb44bb44bb44bb44bb44bb44bb0250bb44bb44bb44bb44bb44bb44bb44bb440
3020468426574612030 2e39a4015033cc33cc33cc33cc33cc33cc33cc33cc0250cc33cc33cc33
cc33cc33cc33cc33cc330300047050726f64756374696f6e20322e352e311904c282a60150001
12233445566778899aabbccddeeff024821e4918cda74da120346a07eba9e8ae3044673733717
71720503064491b8a402a60150112233445566778899aabbccddeeff0002467373371771720 34
7a9301412789124044a02d41e4120db94e1cc2405120648bd9124afe881a991
```

B.1.3 Example Claims Set with Subclaims

```
{
  / Security Rating / 1000 : 5,
  / Cert DLoA / 1300 :
  {
    1: "MyPlatformLabel",
    2: "http://www.dloaregistrar.com"
  },
  / location / 264 :
  {
    / latitude / 1: 34.42874324000000,
    / longitude / 2: -109.9832743250000
  }
```

```
}
```

CBOR encoded value:

```
a31903e805190514a2016f4d79506c6174666f726d4c6162656c02781c687474703a2f2f77777
72e646c6f617265676973747261722e636f6d190108a201fb404136e10ef9172e02fbc05b7eed
f76f37c6
```

B.1.4 Example Claims Set with Submodules

This example uses the following Claims Set for "mySubmodule1":

```
{
  / issuedat / 6: 1444064944
}
```

This example uses the following Claims Set for "mySubmodule2":

```
{
  / issuer / 1: "ACME Corporation",
  / subject/ 2: "CWT Example",
  / audience/ 3: "GlobalPlatform"
}
```

Nested EAT Private Message Key for ECDSA (generated according to [RFC 6979]):

```
f70d0ec2032afad5d0756fde00bb8563aeccee2c43db7e27d0288e296c9aa4e1
```

```
{
  / Security Rating / 1000 : 5,
  / Cert DLoA / 1300 :
  {
    1: "MyPlatformLabel",
    2: "http://www.dloaregistrar.com"
  },
  / submods / 266 :
  {
    "mySubmodule1":
    {
      / iat / 6: 1444064944
    },
    "mySubmodule2":
    61(
      18(
        [
          / protected / h'a10126',
          / unprotected / {
            4: 'signatureKey'
          },
/ payload /
h'a3017041434d4520436f72706f726174696f6e026b435754204578616d706c65036e476
c6f62616c506c6174666f726d',
/ signature /
h'95d3a110f25581f5ea478997772478481e5dc68600514c1191191a3aded63c43d70bf50
0afcdb105aa264f56e57bf88e28b868983b3935b7c168d9d6b12a5df1'
        ]
      )
    )
  }
}
```

CBOR encoded value:

```
a31903e805190514a2016f4d79506c6174666f726d4c6162656c02781c687474703a2f2f77777
72e646c6f617265676973747261722e636f6d19010aa26c6d795375626d6f64756c6531a1061a
5612aeb06c6d795375626d6f64756c6532d83dd28443a10126a1044c7369676e61747572654b6
5795830a3017041434d4520436f72706f726174696f6e026b435754204578616d706c65036e47
6c6f62616c506c6174666f726d584095d3a110f25581f5ea478997772478481e5dc68600514c1
191191a3aded63c43d70bf500afcdb105aa264f56e57bf88e28b868983b3935b7c168d9d6b12a
5df1
```

B.2 Example Unendorsed Claims Sets

All examples in this section use the Claims Set shown below.

```
{
  / Security Rating / 1000 : 5,
  / MUD File URL / 1001 :
"https://mudfile.globalplatform.org/download/example.json",
  / Debug Status / 263 : 3
}
```

B.2.1 Example Unendorsed Claims Set

```
{
  / Security Rating / 1000 : 5,
  / MUD File URL / 1001 :
"https://mudfile.globalplatform.org/download/example.json",
  / Debug Status / 263 : 3
}
```

CBOR encoded value:

```
a31903e8051903e9783868747470733a2f2f6d756466696c652e676c6f62616c706c6174666f7
26d2e6f72672f646f776e6c6f61642f6578616d706c652e6a736f6e19010703
```

B.2.2 Example Tagged Unendorsed Claims Set

```
601 (
  {
    / Security Rating / 1000 : 5,
    / MUD File URL / 1001 :
"https://mudfile.globalplatform.org/download/example.json",
    / Debug Status / 263 : 3
  }
)
```

CBOR encoded value:

```
d90259a31903e8051903e9783868747470733a2f2f6d756466696c652e676c6f62616c706c617
4666f726d2e6f72672f646f776e6c6f61642f6578616d706c652e6a736f6e19010703
```

B.3 Example Entity Attestation Tokens

All examples in this section use the Claims Set shown below.

```
{
  / Security Rating / 1000 : 5,
  / MUD File URL / 1001 :
"https://mudfile.globalplatform.org/download/example.json",
  / Debug Status / 263 : 3
}
```

B.3.1 Example Signed Token

Private Message Key for ECDSA (generated according to [RFC 6979]):

4a7a49ff901026624d75ba06c72907b9ad38e4acebb50d0b2d535b70655d6c27

```
[
  / protected / h'a10126' / {
    1: -7
  } / ,
  / unprotected / {
    4: 'signatureKey'
  },
/ payload /
h'a31903e8051903e9783868747470733a2f2f6d756466696c652e676c6f62616c706c617
4666f726d2e6f72672f646f776e6c6f61642f6578616d706c652e6a736f6e19010703',
/ signature /
h'1c8781dfffc71d78429ead67341dbb4be0ba9fb7750324252b242caa0bb2c0d422fd951
363ae04e2ba3c340bee19c91d91644baa79a540eb6d9c71e23a5231a4'
]
```

CBOR encoded value:

```
8443a10126a1044c7369676e61747572654b65795846a31903e8051903e9783868747470733a2
f2f6d756466696c652e676c6f62616c706c6174666f726d2e6f72672f646f776e6c6f61642f65
78616d706c652e6a736f6e1901070358401c8781dfffc71d78429ead67341dbb4be0ba9fb7750
324252b242caa0bb2c0d422fd951363ae04e2ba3c340bee19c91d91644baa79a540eb6d9c71e2
3a5231a4
```

B.3.2 Example Encrypted Token

```
[
  / protected / h'a1010a' / {
    1: 10
  } / ,
  / unprotected / {
    5: h'd49fd4a4597e35cf3222f4cccf'
  },
/ ciphertext /
h'087c54463077169fc3166a46e101fead07c2eb07c53cb3ab49da8e8e6d3d0cccbc3cfbd
38abd6c81fadf53a3901919c73df507ff4768cb28ad97a45d0f5022d6192061d4e419504c
9e5fee416aa4'
]
```

CBOR encoded value:

```
8343a1010aa1054dd49fd4a4597e35cf3222f4cccf584e087c54463077169fc3166a46e101fea
d07c2eb07c53cb3ab49da8e8e6d3d0cccbc3cfbd38abd6c81fadf53a3901919c73df507ff4768
cb28ad97a45d0f5022d6192061d4e419504c9e5fee416aa4
```

B.3.3 Example MACed Token

```
[
  / protected / h'a10105' / {
    1: 5
  } / ,
  / unprotected / {},
/ payload /
h'a31903e8051903e9783868747470733a2f2f6d756466696c652e676c6f62616c706c617
4666f726d2e6f72672f646f776e6c6f61642f6578616d706c652e6a736f6e19010703',
/ tag /
h'722172794345d5c890afd2911a0a3d0a973d884a0ddfb512682e91f8e0e9012f'
]
```

CBOR encoded value:

```
8443a10105a05846a31903e8051903e9783868747470733a2f2f6d756466696c652e676c6f626
16c706c6174666f726d2e6f72672f646f776e6c6f61642f6578616d706c652e6a736f6e190107
035820722172794345d5c890afd2911a0a3d0a973d884a0ddfb512682e91f8e0e9012f
```

B.3.4 Example Signed Then Encrypted Token

Private Message Key for ECDSA (generated according to [RFC 6979]):

4a7a49ff901026624d75ba06c72907b9ad38e4acebb50d0b2d535b70655d6c27

```
[
  / protected / h'a1010a' / {
    1: 10
  } / ,
  / unprotected / {
    5: h'd49fd4a4597e35cf3222f4cccf'
  },
/ ciphertext /
h'0a7c56a494045b13c85a6756ca34ccc3f0d054f6e2fbdbe18df0a5916b3803c1b825e8c
2a7b767b4921338c757786fb3bae150f65733de23f6d5fb5c1f1a2ecc1a2b56b28f759744
eaec17b3cca2818f86634f050835c174496ff375302d8465837d92c0342a2595beb61b27b
08d8aaf10c69a7f95aef25f6bf467bf23d07c196db9e7de6c6931a061b66f21db5aa817da
83890e185614d1afb7e17ed3578b595179185320a2630e8f56c2516885d29710d24e463ff
c0a73ec'
]
```

CBOR encoded value:

```
8343a1010aa1054dd49fd4a4597e35cf3222f4cccf58b90a7c56a494045b13c85a6756ca34ccc
3f0d054f6e2fbdbe18df0a5916b3803c1b825e8c2a7b767b4921338c757786fb3bae150f65733
de23f6d5fb5c1f1a2ecc1a2b56b28f759744eaec17b3cca2818f86634f050835c174496ff3753
02d8465837d92c0342a2595beb61b27b08d8aaf10c69a7f95aef25f6bf467bf23d07c196db9e7
de6c6931a061b66f21db5aa817da83890e185614d1afb7e17ed3578b595179185320a2630e8f5
6c2516885d29710d24e463ffc0a73ec
```

B.3.5 Example Tagged Encrypted Token

This example is the same as example B.3.2 except that it adds the CBOR Tag that indicates an Entity Attestation Token, and thus also the CBOR Tag that indicates a `COSE_Encrypt0` message.

```
61(
  16(
    [
      / protected / h'a1010a' / {
        1: 10
      } / ,
      / unprotected / {
        5: h'd49fd4a4597e35cf3222f4cccf'
      },
/ ciphertext /
h'087c54463077169fc3166a46e101fead07c2eb07c53cb3ab49da8e8e6d3d0cccbc3cfbd
38abd6c81fadf53a3901919c73df507ff4768cb28ad97a45d0f5022d6192061d4e419504c
9e5fee416aa4'
    ]
  )
)
```

CBOR encoded value:

```
d83dd08343a1010aa1054dd49fd4a4597e35cf3222f4cccf584e087c54463077169fc3166a46e
101fead07c2eb07c53cb3ab49da8e8e6d3d0cccbc3cfbd38abd6c81fadf53a3901919c73df507
ff4768cb28ad97a45d0f5022d6192061d4e419504c9e5fee416aa4
```