TCG EK Credential Profile
For TPM Family 2.0; Level 0

Version 2.3
Revision 2
23 July 2020

Contact: admin@trustedcomputinggroup.org

PUBLISHED

TCG

Disclaimers, Notices, and License Terms

Acknowledgement

The TCG wishes to thank those who contributed to this specification. This document builds on considerable work done in the various working groups in the TCG.

Special thanks to the members of the IWG group and others contributing to this document:

Name	Member Company
Carolin Baumgartner (Group Chair)	Carolin Baumgartner
Bob Bell	Cisco Systems
Max Pritikin	Cisco Systems
Monty Wiseman (Group Chair)	General Electric Company
Tom Laffey	Hewlett Packard Enterprise
Scott Kelly	Hyperthought
Ken Goldman	IBM
Ga-Wai Chin (Editor)	Infineon
Georg Rankl	Infineon
Stefan Kaeser	Infineon
Eduardo Cabre	Intel Corporation
David Challener	Johns Hopkins University, Applied Physics Lab
Gabriel Stocco	Microsoft
Rahul Verma	Microsoft
Ronald Aigner	Microsoft
Will Arthur	Raytheon Cyber Solutions, Inc
Olivier Collart	STMicroelectronics
Gloria Serrao	United States Government
Greg Kazmierczak	Wave Systems

Table of Contents

1 Introduction 7
1.1 Purpose 7
1.2 Scope 7
1.3 Relationship to Other TCG Specifications 7
1.4 Keywords 7
1.5 Abbreviations 7
1.6 Definition of Terms 7
2 TPM 2.0 EK and EK Credential 8
2.1 Endorsement Key 8
2.1.1 Primary Key Generation 8
2.1.2 EK Usage 8
2.1.2.1 User Device TPM 9
2.1.2.2 Non-User Device TPM 9
2.1.3 EK Lifetime 9
2.2 Endorsement Key Credential 11
2.2.1 NV Index Handles 11
2.2.1.1 General Design 11
2.2.1.2 NV Index Contents 11
2.2.1.3 Allowed and Recommended Usages of NV Indices 12
2.2.1.4 Low Range 12
2.2.1.5 High Range 13
2.2.1.6 TPMT_PUBLIC Calculation 15
2.2.1.7 Locating Keys or specific NV Index Content 15
2.2.1.8 Key Handle and Certificate Handle Relationships 15
2.2.1.9 Read EK certificates and create the associated EKs 15
2.2.2 EK Credential Lifetime 17
2.3 Privacy Protection 18
3 X.509 ASN.1 Definitions 19
3.1 TCG Attributes 19
3.1.1 TPM Security Assertions 19
3.1.2 TPM Device Attributes 22
3.1.3 TPM Specification Attributes 22
3.2 EK Certificate 23
3.2.1 Version 24
3.2.2 Serial Number 24
3.2.3 Signature Algorithm 24
3.2.4 Issuer 24
3.2.5 Validity 24
3.2.6 Subject 24
3.2.7 Subject Public Key Info 25
3.2.8 Certificate Policies 25
3.2.9 Subject Alternative Name 25
3.2.10 Basic Constraints 25
3.2.11 Subject Directory Attributes 25
3.2.12 Authority Key Identifier 26
3.2.13 Authority Information Access 26
3.2.14 CRL Distribution 26
3.2.15 Key Usage 26
3.2.16 Extended Key Usage 27
3.2.17 Subject Key Identifier 27
4 X.509 ASN.1 Structures and OIDs 28
5 References 31

A. Certificate Examples 33
A.1 Example 1 (user device TPM, e.g. PC-Client) 33
B. Default EK Templates (algorithm-specific) 35
B.1 Introduction 35
B.2 Backwards Compatibility 35
B.3 EK Templates in the Low Range 35
B.3.1 Introduction 35
B.3.2 Satisfying *PolicyA* 36
B.3.3 Template L-1: RSA 2048 (Storage) 37
B.3.4 Template L-2: ECC NIST P256 (Storage) 38
B.4 EK Templates in the High Range 39
B.4.1 Introduction 39
B.4.2 Authorization Options 39
B.4.3 Satisfying *PolicyB* 39
B.4.4 Template H-1: RSA 2048 (Storage) 41
B.4.5 Template H-2: ECC NIST P256 (Storage) 42
B.4.6 Template H-3: ECC NIST P384 (Storage) 43
B.4.7 Template H-4: ECC NIST P521 (Storage) 44
B.4.8 Template H-5: ECC SM2 P256 (Storage) 45
B.4.9 Template H-6: RSA 3072 (Storage) 46
B.4.10 Template H-7: RSA 4096 (Storage) 47
B.5 Policy NV Indices 48
B.5.1 Introduction 48
B.5.2 Handle Values 49
B.5.3 Policy Index I-1: SHA256 50
B.5.4 Policy Index I-2: SHA384 51
B.5.5 Policy Index I-3: SHA512 52
B.5.6 Policy Index I-4: SM3_256 53
B.6 Policy Computation 54
B.6.1 Introduction 54
B.6.2 Computing PolicyA 54
B.6.3 Computing Policy Index Names 55
B.6.4 Computing PolicyC 55
B.6.5 Computing PolicyB 56
C. Certificate Fields (algorithm-specific) 57
C.1 Signature Algorithm 57
C.1.1 RSA 57
C.1.1.1 RSA 2k CA Key 57
C.1.1.2 RSA 3k and 4k CA Key 57
C.1.2 ECC 57
C.1.2.1 NIST P256 CA Key 57
C.1.2.2 NIST P384 CA Key 57
C.1.2.3 NIST P521 CA Key 57
C.1.2.4 SM2 P256 CA Key 57
C.2 Subject Public Key Info 58
C.2.1 RSA 58
C.2.2 ECC 58
C.2.2.1 NIST P256 58
C.2.2.2 NIST P384 58
C.2.2.3 NIST P521 58
C.2.2.4 SM2 P256 59

Tables

Table 1: EK Certificate Fields 24
Table 2: Default EK Template (TPMT_PUBLIC) L-1: RSA 2048 (Storage) 37
Table 3: Default EK Template (TPMT_PUBLIC) L-2: ECC NIST P256 (Storage) 38
Table 4: Default EK Template (TPMT_PUBLIC) H-1: RSA 2048 (Storage) 41
Table 5: Default EK Template (TPMT_PUBLIC) H-2: ECC NIST P256 (Storage) 42
Table 6: Default EK Template (TPMT_PUBLIC) H-3: ECC NIST P384 (Storage) 43
Table 7: Default EK Template (TPMT_PUBLIC) H-4: ECC NIST P521 (Storage) 44
Table 8: Default EK Template (TPMT_PUBLIC) H-5: SM2 P256 (Storage) 45
Table 9: Default EK Template (TPMT_PUBLIC) H-6: RSA 3072 (Storage) 46
Table 10: Default EK Template (TPMT_PUBLIC) H-7: RSA 4096 (Storage) 47
Table 11: EK Policy Index (TPMS_NV_PUBLIC) I-1: SHA256 50
Table 12: EK Policy Index (TPMS_NV_PUBLIC) I-2: SHA384 51
Table 13: EK Policy Index (TPMS_NV_PUBLIC) I-3: SHA512 52
Table 14: EK Policy Index (TPMS_NV_PUBLIC) I-4: SM3_256 53
Table 15: PolicyA values 54
Table 16: Policy Index Names 55
Table 17: PolicyC values 56
Table 18: PolicyB values 56

Figures

Figure 1: Overview EK Template to Policy Index 48

1 Introduction

1.1 Purpose

The purpose of this document is to define the TPM 2.0 Endorsement Key (EK) Credential. This specification describes the content of the credential and provides an X.509 instantiation of the credential. A standardized and commonly used format should provide better interoperability between credential providers and users.

1.2 Scope

This document specifies the TPM 2.0 Endorsement Key Credential. It does not apply to TPM 1.2 credentials or credentials of other type.

1.3 Relationship to Other TCG Specifications

A TPM claiming adherence to this specification SHALL be compliant with the TPM 2.0 Library Specification[1]; Family 2.0; Level 00; Revision 00.99 or later.

1.4 Keywords

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119[17].

1.5 Abbreviations

CFB Cipher Feedback mode

CSR Certificate Signing Request

EK Endorsement Key

EPS Endorsement Primary Seed

IDevID Initial Device Identifier

KDF Key Derivation Function

OEM Original Equipment Manufacturer

RDN Relative Distinguished Name

TPM2_ Prefix that indicates a TPM 2.0 command

1.6 Definition of Terms

The TCG Glossary [20] contains a few definitions that are fundamental to this document.

The following operational definitions, however, are specific to this specification.

Certificate – A certificate is an instantiation of a credential using the industry-standard certificate structure from ISO/IEC/ITU-T X.509 version 3. Certificate generation consists of (a) assembling values for the credential fields and (b) signing over the assembled fields.

Credential – A credential is an abstract proof that must be instantiated as a certificate before it can be exchanged between entities.

2 TPM 2.0 EK and EK Credential

2.1 Endorsement Key

The Endorsement Key (EK) is an asymmetric key pair consisting of a public and private key stored in a Shielded Location on the TPM. The public part of the EK can be read from the TPM while the private part MUST never be exposed. The public key of the EK is included in the EK certificate.

In TPM 1.2, the Endorsement Key was defined as an RSA 2048 bit key. This is not the case for TPM 2.0, which can have more than one EK. The algorithm flexibility provided by the TPM 2.0 Library Specification [1] allows the TPM to create EKs of any type of asymmetric algorithm implemented in the TPM (see 2.1.1). The properties of the Endorsement Key are defined by its public area structure, the “template”. This specification defines multiple default templates for algorithm-specific Endorsement Keys.

NOTE The default templates are defined in an annex of this document to allow easier integration of additional default templates, which will be provided as they become available.

Any asymmetric algorithm supported in a platform-specific specification used to implement the TPM MAY be used as a key type instead of or in addition to the defined key types in the annex of this document.

A relevant Platform specification may provide guidance as to whether or not the EK should be persistent in the TPM when it ships.

2.1.1 Primary Key Generation

The Endorsement Key is a Primary Object controlled by the Endorsement Hierarchy. The Endorsement Hierarchy has a Seed, the Endorsement Primary Seed (EPS) which is unique to each TPM. The Primary Seed is a large random value; its size is required to be at least twice the security strength of any algorithm implemented on the TPM. The EPS MUST be generated within the TPM or MUST be generated and injected by the TPM manufacturer in the manufacturing environment. The attribute TPMA_PERMANENT.tpmGeneratedEPS (see *TPM 2.0 Library Specification, Part 2[1]*) MUST be set properly to indicate the source of the Seed. The attribute can be read with TPM2_GetCapability(). The Seed cannot be read from the TPM and MUST never be exposed.

The TPM 2.0 Library Specification [1] defines a process to create primary keys based on a Key Derivation Function (KDF) and a Primary Seed. The KDF is a deterministic function that uses key parameters to derive a reproducible key. These parameters determine the type of the key and are input to the command TPM2_CreatePrimary(). When this command is called with the same parameters, the same key is generated as long as the EPS does not change. The key can be made persistent in TPM NV memory using the command TPM2_EvictControl() or recreated when needed. This way, any type of key (e.g. symmetric or asymmetric, signing or decryption key) can be created by the TPM.

As of revision 1.38 of the TPM 2.0 Library Specification [1], TPM2_CreatePrimary() uses the sensitive data (provided in the *inSensitive.sensitive.data* parameter) as part of the calculation for the Primary Key. In the creation of an Endorsement Key for which an EK Credential is issued, the sensitive data size (*inSensitive.sensitive.data.t.size*) MUST be set to zero.

2.1.2 EK Usage

In TPM 1.2, the Endorsement Key was defined as a decryption key; it could not be used for signing operations. Unlike TPM 1.2, TPM 2.0 provides more flexibility in defining an EK. The properties of the Endorsement Key are determined by its public area template (TPMT_PUBLIC structure, see *TPM 2.0 Library Specification Part 2 [1]*). The TPMT_PUBLIC structure includes the base attributes restricted, sign and decrypt that determine the cryptographic operation a key may perform on an object. The TPM 2.0 Library Specification [1] does not impose any restrictions regarding the attributes of the Endorsement Key. As any other key the Endorsement Key can be a created as a decryption or signing key.

However, the EK and its credential may be considered privacy-sensitive if the private part of the EK is used in a cryptographic protocol. In this case, the public EK or the EK certificate may represent a privacy-sensitive cryptographic identifier for a particular platform. In privacy-sensitive environments, the EK SHOULD NOT be used as a signing key and restricted to specific operations (this is described in more detail in section 2.3 Privacy Protection).

On the other hand, there are environments where privacy is not an issue. This specification distinguishes between user device TPMs and non-user device TPMs. Whether the EK may sign depends on the type of platform for which the TPM is built.

2.1.2.1 User Device TPM

User device TPMs are TPMs that are associated with a human user, typically in PC Client or Mobile platforms. If the EK is certified by a trusted entity, the EK SHOULD NOT be used for signing operations due to privacy concerns. In this case, the EK SHOULD be defined as a restricted decryption (Storage) key.

2.1.2.2 Non-User Device TPM

Non-user device TPMs on the other hand are TPMs that are associated with an enterprise, rather than a specific user. This can be e.g.

- Network Elements (e.g. routers, switches, wireless access points)
- Servers, Virtual Servers, Virtual Devices in a cloud infrastructure
- Embedded Devices (e.g. printers)

For such platforms privacy is not a central concern and unique identification is of critical importance. These platforms MAY use a certified EK for signing operations. This is intended to facilitate establishment of further TPM keys, like Device Identification keys, without the need for an Attestation Key. This allows for simpler infrastructure implementations. In this case, no restrictions other than those defined in the TPM 2.0 Library Specification [1] apply to the settings of the base attributes. The EK MAY be defined as a general-purpose key if both signing and decryption should be supported. The key usage field in the EK Credential defined in section 3.2.15 MUST be set appropriately to indicate the usage of the EK.

One use case for a signing EK is to sign the Certificate Signing Request (CSR) for an Initial Device Identifier (IDevID) key. The IDevID key is a TPM-generated key that is used as an initial identity for secure device authentication (see IEEE 802.1AR [10]). The CSR can be signed with the command TPM2_Sign(). The hash calculated over the certification request information is passed to the TPM in the digest command parameter; the inScheme command parameter specifies the signing scheme. The issuer of the IDevID certificate could verify the CSR signature to ensure that the chip requesting the IDevID certificate is privileged to receive it. Therefore the issuer could have a list of EK certificates of all valid TPMs a product manufacturer has purchased. Alternatively, a CSR could be signed by an Attestation Key.

2.1.3 EK Lifetime

In TPM 2.0, the lifetime of an Endorsement Key is tied to the Endorsement Primary Seed (EPS). As long as the EPS is not changed, EKs can be recreated with their public area templates. The command TPM2_ChangeEPS() replaces the Endorsement Primary Seed with a new random value and makes it impossible to recreate any EKs derived from the previous Seed. This will invalidate all certificates associated with the EKs.

Platform-specific specifications determine whether the command TPM2_ChangeEPS() is required to be implemented. Some platforms might want to change the EPS, e.g. during platform refurbishment to erase existing EKs or after a field upgrade from a firmware that had a severe security flaw (in order to revoke all EKs associated with the old firmware).

On the other hand, there are platforms that need a permanent EPS because invalidating the Endorsement Keys would prevent the platform proving that it is a genuine trusted platform. In non-

user device TPMs (see 2.1.2.2), for instance, the EPS is required to be permanent because the EK represents the trust anchor for the device identity.

The TPM 2.0 Library Specification [1] provides a means to prevent the EKs from being replaced. The command to change the EPS requires Platform Authorization, so the OEM can decide if the EPS ever changes. The use of Platform Authorization can be disallowed by turning off the Platform hierarchy (by setting the phEnable flag to CLEAR); this disables any functionality in the TPM that would require platformAuth or platformPolicy. Furthermore, TPM2_ChangeEPS() can be added to the list of commands that require assertion of Physical Presence using TPM2_PP_Commands(). Alternatively, platforms could prevent EKs from being erased by not exposing this functionality to the user.

2.2 Endorsement Key Credential

The Endorsement Key Credential is an X.509 v3 certificate that contains the public EK, as well as various assertions regarding the security qualities and provenance of the TPM. The definition of the certificate fields are specified in section 3.2. The EK Credential is usually issued by a TPM or Platform manufacturer during manufacturing process. An entity SHALL NOT create an EK Credential for a TPM unless the entity is satisfied that the public key referenced in the EK Credential was either:

- returned in response to a TPM2_CreatePrimary() command by an implementation of Protected Capabilities and Shielded Locations that meets the TPM 2.0 Library Specification [1] or
- generated outside the TPM and inserted by a process defined in the Target of Evaluation (TOE) of the security target in use to evaluate the TPM.

There might be use cases where it is useful to issue an EK Credential after manufacturing (e.g. if the EPS was changed or the TPM is shipped without EK). In this case, the entity issuing the credential would create a new Endorsement Key with TPM2_CreatePrimary(). This procedure would require support for certificate enrollment. Support for an enrollment protocol is optional and MAY be done using a proprietary method of the TPM or Platform manufacturer or a method standardized by TCG. One example implementation (at time of writing, only available for TPM 1.2) is described in the IWG document *CMC Profile for EK/Platform Certificate Enrollment [7]*.

A primary use case of an EK Credential is to assist Attestation CAs to issue credentials for restricted signing keys (Attestation Keys). The EK Credential can be used to provide evidence that the Attestation Keys are resident on the same TPM as the EK.

In TPM 2.0, multiple EKs can be derived from a single Seed (as described in section 2.1.1.) As a result, the TPM can have more than one EK Credential. However, the TPM might not be provisioned with all the credentials because of NV space restrictions; the credentials could be stored off the TPM. If an EK Credential is stored on the TPM, it is stored as an NV Index; in this case, it is referenced by its NV Index handle (see 2.2.1). The authorization to modify or access (read, write, delete) the credential is determined by its attributes. The attributes of the Index are defined by platform-specific workgroups, as well as the authorization for the index. Definitions specific to PC Client can be found in *PC Client Specific Platform TPM Profile for TPM 2.0 [6],* section Non-volatile Storage.

2.2.1 NV Index Handles

2.2.1.1 General Design

The NV Index handles related to the EK Credential MUST have values that are defined by the TCG in the *Registry of Reserved TPM 2.0 Handles and Localities [2]*. This section uses handles of the type "Global NV indices for OEMs, assigned by TCG", Refined Handle Type Endorsement certificate.

This section defines two variations. The Low Range (see 2.2.1.4) generally includes the RSA 2048 and ECC NIST P256 EK data. The High Range (see 2.2.1.5) generally includes EK data for other algorithms. Both ranges permit the inclusion of an EK Template. In addition, the Low Range permits inclusion of a distinct EK Nonce. The High Range does not permit a distinct EK Nonce.

This document uses the following terms with specific meanings:

Term	Specific meaning within this document
Absent	This NV Index is not defined (i.e., this NV Index does not exist)
Populated	This NV Index is defined and written with content relevant to this document.

Within this document, manufacturer means either the TPM manufacturer or Platform manufacturer.

2.2.1.2 NV Index Contents

This section defines the NV Index contents, and is applicable to both the Low Range and High Range.

NOTE: There is no length, type, or other metadata stored in the NV Index data.

2.2.1.2.1 EK Certificate

An EK Certificate is stored in an NV Index as an X.509 certificate encoded in DER format. The NV Index data contains only the DER certificate data.

2.2.1.2.2 EK Template

An EK Template is stored in an NV Index as a TPMT_PUBLIC structure marshaled as described in the TPM 2.0 Library Specification [1]. The default EK Templates are defined in annex B. The EK Template NV Index MUST be *Populated* if non-default values are used. It SHOULD be *Absent* if default values are used.

The EK Template unique field buffer size(s) SHOULD be zero.

NOTE 1: Setting the unique field buffer size(s) to zero minimizes the use of NV space. The unique field is only necessary to generate multiple different EKs if the rest of the TPMT_PUBLIC is the same. If other portions of the TPMT_PUBLIC change (e.g., the authPolicy), a different EK will be generated even if the unique field size(s) is zero.

NOTE 2: A platform-specific working group may define a proprietary default EK public area template that can be used instead of the templates defined in annex B.

2.2.1.2.3 EK Nonce

An EK Nonce is stored in an NV Index and is used to modify the EK Template unique field. The EK Nonce size is determined by the size field of the NV Index public area.

2.2.1.3 Allowed and Recommended Usages of NV Indices

This table provides an overview of the allowed configurations of EK Nonce and EK Template in the Low and High ranges.

EK Nonce	EK Template	Allowed in Low Range	Allowed in High Range
NV index *Absent*	NV index *Absent*	SHOULD	SHOULD
NV index *Absent*	NV index *Populated*	SHOULD NOT	See 2.2.1.5
NV index *Populated*	NV index *Absent*	MUST NOT	MUST NOT
NV index *Populated*	NV index *Populated*	SHOULD NOT	MUST NOT

2.2.1.4 Low Range

The Low Range is at NV Indices 0x01c00002 - 0x01c0000c. For TPMs designed to meet Windows [22], the low range MUST be used for the RSA 2048 EK and the ECC NIST P256 EK.

NOTE 1: The Windows OS uses NV Indices in this range:

0x01c00002 RSA 2048 EK Certificate

0x01c00003 RSA 2048 EK Nonce

0x01c00004 RSA 2048 EK Template

0x01c0000a ECC NIST P256 EK Certificate

0x01c0000b ECC NIST P256 EK Nonce

0x01c0000c ECC NIST P256 EK Template

EK Certificate NV Index(es) SHOULD be *Populated* by the manufacturer.

The manufacturer SHOULD leave the EK Nonce NV Index *Absent*. If a unique field is specified, it SHOULD be included as part of the associated EK Template NV Index.

NOTE 2: The preferred provisioning uses default EK Templates to conserve TPM NV space.

2.2.1.5 High Range

The High Range is at 0x01c00012 and upwards. For TPMs designed to meet Windows [22], the high range MUST be used for an EK other than the first RSA 2048 and ECC NIST P256 keys.

For TPMs designed to meet Windows [22], the High Range can be used for additional RSA 2048 or ECC NIST P256 keys, but the first RSA 2048 and ECC NIST P256 key MUST be provisioned in the Low Range.

Any *Populated* even index MUST contain an EK certificate. For any EK Certificate, an EK Template MAY be included. If included it MUST be *Populated* in the next (subsequent odd) index.

EK Nonces SHALL NOT be *Populated* in any NV Index in the High Range.

NOTE 1: There is no need to tightly pack the certificate / template pairs. Any NV Index in the range permitted in the *Registry of Reserved TPM 2.0 Handles and Localities [2]* is acceptable.

If the TCG defines a default template for the EK certificate, the EK Template SHOULD be *Absent*. If the TCG does not define a default template for the EK certificate, the EK Template MUST be *Populated*.

NOTE 2: The preferred provisioning uses default EK Templates to conserve TPM NV space.

2.2.1.5.1 Handle Values for EK Certificates

If EK Certificates are *populated*, then the following list of NV Index handles SHALL be used to store the EK Certificates corresponding to the EKs created with the default Templates defined in the High Range.

NOTE 1: Platform profiles define which EK certificates must be *populated*.

NOTE 2: The handle values are normative as of version 2.3 of this specification. Defining the NV index handles as normative simplifies EK certificate validation because it eliminates parsing of certificate content in order to identify the algorithm (RSA or ECC) and key/curve size.

0x01c00012	RSA 2048 EK Certificate (H-1)
0x01c00014	ECC NIST P256 EK Certificate (H-2)
0x01c00016	ECC NIST P384 EK Certificate (H-3)
0x01c00018	ECC NIST P521 EK Certificate (H-4)
0x01c0001a	ECC SM2_P256 EK Certificate (H-5)
0x01c0001c	RSA 3072 EK Certificate (H-6)
0x01c0001e	RSA 4096 EK Certificate (H-7)

2.2.1.5.2 Handle Values for EK Certificate Chains

Storing EK Certificate chains in NVRAM is optional and generally not recommended due to potentially large NV space consumption. Nonetheless, there may be scenarios where provisioning the EK certificate chain in the TPM is necessary, such as when standard EK Certificate Chain distribution methods are unavailable. If the TPM manufacturer decides to provision the EK Certificate Chain in NV, index handles and formats defined in this section SHOULD be used.

TPM NV MAY contain all certificates of the EK Certificate Chain except the Root CA certificate. The EK Certificate Chain MUST be stored as X.509 DER encoded certificates. If the chain consists of more than one certificate, or if multiple chains exist, they MUST be stored in NV as a concatenated sequence. The TPM manufacturer MAY provision certificate chains using the following list of indices:

0x01c00100	EK Certificate Chain Index 1
…	
0x01c001ff	EK Certificate Chain Index 256

The NV indices MUST be populated starting with index 0x01c00100 through index 0x01c001ff. There is no requirement to store the certificates in any particular order whatsoever. If a concatenated certificate chain does not fit in a single index, the chain MUST overflow to the next numerically larger index in the list of NV Indices. If the storage space in a single index is insufficient to store the entire certificate, the certificate MAY overflow into the next numerically larger index in the list of NV Indices. It is recommended to use the least number of indices possible for storing the chains.

If more than one EK Certificate Chain exists, the chains MUST be concatenated. If two or more chains have common certificates, such as when they are anchored to the same intermediate or root CA, the certificates MUST NOT be stored more than once. Verifiers are recommended to read the NV Indices in order and store a copy of the contents (certificates) into a memory buffer. Individual certificates may then be parsed from the buffer into a certificate store in order to perform EK Certificate chain validation.

Example 1: a TPM is comprised of the following two EK Certificates chains:

1. ECC Root CA -> ECC Intermediate CA 1 -> ECC Issuing CA -> ECC EK Certificate (leaf)
2. RSA Root CA -> RSA Intermediate CA 1 -> RSA Intermediate CA 2 -> RSA Issuing CA -> RSA EK Certificate (leaf)

The EK Certificate Chain NV Indices may be provisioned as follows:

NV Index	Content (… overflow, || concatenation)
0x01c00100	ECC Issuing CA || ECC Intermediate CA 1 …
0x01c00101	… ECC Intermediate CA 1 || RSA Issuing CA || RSA Intermediate CA 2 …
0x01c00102	… RSA Intermediate CA 2 || RSA Intermediate CA 1
0x01c00103	(undefined)

In Example 1, the certificate ECC Intermediate CA 1 is too long to store in index 0x01c00100, therefore it overflows into the immediately following index (0x01c00101). The RSA chain is stored immediately following the ECC certificate chain, starting at index 0x01c00101.

Example 2: a TPM is comprised of the following two EK Certificates chains which share the same root and intermediate CA:

1. ECC Root CA -> ECC Intermediate CA -> ECC Issuing CA 1 -> ECC EK Certificate (leaf)
2. ECC Root CA -> ECC Intermediate CA -> ECC Issuing CA 2 -> ECC EK Certificate (leaf)

The EK Certificate Chain NV Indices may be provisioned as follows:

NV Index	Index Content (… overflow, || concatenation)
0x01c00100	ECC Issuing CA 1 || ECC Intermediate CA …
0x01c00101	… ECC Intermediate CA || ECC Issuing CA 2
0x01c00102	(undefined)

In Example 2, the ECC Intermediate CA certificate is stored only once in NV memory, since the certificate chains share the same intermediate CA.

NOTE: It is the responsibility of the TPM vendor to provide mechanisms to perform PKI certificate chain validation. This may involve providing revocation information through a CRL Distribution Point or OCSP responder. This specification does not prescribe mechanisms for CA compromise recovery.

2.2.1.6 TPMT_PUBLIC Calculation

The TPMT_PUBLIC structure forms part of the input to the TPM2_CreatePrimary() command.

- If the EK Template is *Absent*, the default template is used as the TPMT_PUBLIC.
- If the EK Template is *Populated* and the EK Nonce is *Absent*, the EK Template is used unmodified as the TPMT_PUBLIC.
- The case of an EK Template *Absent* and an EK Nonce *Populated* is unspecified and MUST NOT be provisioned.
- If the EK Template is *Populated* and the EK Nonce is *Populated*, form the TPMT_PUBLIC as follows:
    1. Begin with the EK Template.
    2. Add the EK Nonce to the default template as follows:
        - For RSA 2048, the EK Nonce is padded to 256 bytes by appending 0x00 bytes. This value is inserted into the default template unique.rsa.t.buffer, and unique.rsa.t.size is set to 256.
        - For ECC NIST P256, the EK Nonce is padded to 32 bytes by appending 0x00 bytes. This value is inserted into the default template unique.ecc.x.t.buffer, and unique.ecc.x.t.size is set to 32. The unique.ecc.y.t.buffer is set to 32 0x00 bytes and the unique.y.size is set to 32.

2.2.1.7 Locating Keys or specific NV Index Content

NOTE: This section is informative. It is provided for convenience only.

Persistent Keys and NV Index content typically do not change between platform reset cycles. If locating these entities during a boot cycle is resource sensitive, software should locate Persistent Keys or NV Index content during initial installation or provisioning and store the specific location on the platform.

Discovery of a Persistent Key can be done by TPM2_GetCapability() - TPM_CAP_HANDLES - TPM_HT_PERSISTENT.

Discovery of the NV Index content (for example, a certificate) can be done by TPM2_GetCapability() - TPM_CAP_HANDLES - TPM_HT_NV_INDEX.

2.2.1.8 Key Handle and Certificate Handle Relationships

Unlike TPM 1.2, TPM 2.0 does not require persistent Endorsement Keys. They can be repeatedly created as transient keys on demand, while a persistent EK would consume scarce NV space.

If an EK is made persistent, it may be easier for software if there is a relationship between the EK persistent handle and the EK certificate NV Index. For example, if an Endorsement Certificate within the Endorsement Certificate range in Table “Reserved Handles for NV indices” of [2] has an Endorsement Primary Key within Table “Key Handles for Persistent Objects” of [2] the offset of each entity could be the same within each respective range. For example, an Endorsement Certificate at NV Index 0x01C00022 (offset 0x22 starting from the beginning of the assigned NV Index range) could have an Endorsement Primary Key at handle 0x81010022 (offset 0x22 starting from the beginning of the assigned key handle range).

2.2.1.9 Read EK certificates and create the associated EKs

The following describes a high level procedure how to read the EK certificates from the TPM and how to create the associated Endorsement Keys. Differences in the Low and High Range are pointed out.

1. Get a list of all NV indices stored in the handle range reserved for EK certificates (0x01C00000 – 0x01C07FFF) using TPM2_GetCapabilty(). The handle range is defined in the Registry of Reserved TPM 2.0 Handles and Localities [2].

2. Identify whether the returned NV index handles lie in the Low Range (0x01C00002 - 0x01C0000C) or in the High Range (0x01C00012 - 0x01C07FFF).
    a. In the Low Range, an EK Certificate, an EK Nonce (recommended to be *Absent*), and an EK Template (recommended to be *Absent*) are *Populated* at assigned standard handle values. If present,
        i. an EK Certificate is at 0x01c00002 (RSA) or 0x01c0000a (ECC)
        ii. an EK Nonce is at 0x01c00003 (RSA) or 0x01c0000b (ECC)
        iii. an EK Template is at 0x01c00004 (RSA) or 0x01c0000c (ECC)
    b. In the High Range, no standard handle values are assigned. An EK Certificate is *Populated* at an even handle value, and (if present) an EK Template is *Populated* at the subsequent odd handle value.
3. Read all NV index handles or those of interest returned in the list from step 1 using TPM2_NV_ReadPublic(), and TPM2_NV_Read().
4. Identify the type of EK certificate in order to create the associated EK.
    a. In the Low Range, the certificate *Populated* at
        i. 0x01c00002 is an RSA 2048 certificate
        ii. 0x01c0000a is an ECC NIST P256 certificate
    b. In the High Range, it is necessary to parse the content of the certificate because no standard handle values are assigned. Parse the SubjectPublicKeyInfo field in the certificate to determine the algorithm and key size/curve - If the algorithms is
        i. rsaEncryption (OID 1 2 840 113549 1 1 1), it is an RSA certificate
            1. This document only defines a Template for 2048, so it is an RSA 2048 bit key
        ii. ecPublicKey (OID 1 2 840 10045 2 1), it is an ECC certificate
            1. Parse namedCurve in the ECParameter to determine the curve – if namedCurve is
                a. secp256r1 (OID 1 2 840 10045 3 1 7), it is a NIST P256 key
                b. secp384r1 (OID 1 3 132 0 34), it is a NIST P384 key
                c. secp521r1 (OID 1 3 132 0 35), it is a NIST P521 key
                d. SM2EllipticCurveCryptography (OID 1 2 156 10197 1 301), it is an SM2 P256 key
5. Create the associated EK using TPM2_CreatePrimary() (typically with a NULL password)
    a. Call TPM2_CreatePrimary() with
        i. *inPublic.publicArea* set to
            1. If the EK Template and/or the EK Nonce is *Populated,* the parameter values defined in section 2.2.1.5.2
            2. Otherwise the parameter values of default EK Templates defined in annex B of this document
        ii. *inSensitive.sensitive.data* set to Empty Buffer (see 2.1.1)

b. Parse the public key from the returned *outPublic.publicArea.unique* parameter and compare it with the public key stored in the certificate. If they match, the key corresponds to the certificate.

2.2.2 EK Credential Lifetime

An EK Credential contains fields that express the validity period of the credential. The validity period is at the discretion of the manufacturer. The credential is not expected to expire during the normal life expectancy of the platform in which it resides. The lifetime can vary widely between different types of platforms (e.g. while a typical validity period for a PC Client platform is 5-10 years, non-user device TPMs are expected to operate indefinitely into the future in which case the value 99991231235959Z should be used as expiration date). The credential lifetime can also depend on the lifetime of the TPM device and the algorithm type of the Endorsement Key. The time frame during which the security strength of the EK is acceptable SHOULD be taken into account by the manufacturer when determining the credential lifetime (e.g. see SP800-57 [11]).

However, an EK Credential can become useless before expiration of the validity period if the associated EK is irrevocably erased from the TPM. This is the case if the EPS is replaced (see 2.1.3 EK Lifetime).

TPM 2.0 provides functionality to define a permanent NV Index. In TPM 2.0, the NV Index attributes TPMA_NV_PLATFORMCREATE and TPMA_NV_POLICY_DELETE (see TPM *2.0 Library Specification, Part 2 [1]*) determine the authorization required to delete an NV Index. If such an NV Index is created such that Platform Authorization is required to write it, the EK Credential can be protected against accidental deletion (e.g. by the Owner).

TPMA_NV_PLATFORMCREATE indicates whether the NV Index was defined by the platform. If SET, the index may only be undefined with Platform Authorization and not with Owner Authorization. TPMA_NV_PLATFORMCREATE SHOULD be SET for an EK Credential to prevent the credential from being deleted if the Owner is cleared. Platform-defined NV indices can also SET TPMA_NV_POLICY_DELETE.

If TPMA_NV_POLICY_DELETE is SET, the Index cannot be deleted unless the authPolicy of the NV Index is satisfied using the command TPM2_NV_UndefineSpaceSpecial(). Similarly it is possible to create an NV index that cannot be written without a policy authorization. A platform that requires a permanent EK Credential would not create a policy that allows the EK Credential to be removed or overwritten. On the other hand, a platform that wants to clear the EK Credential, e.g. during platform refurbishment, could create a policy that includes among the AND terms the command TPM2_PolicyCommandCode() where the command code is set to TPM_CC_NV_UndefineSpaceSpecial along with policy commands that control the authorization.

The settings of the NV Index attributes are determined by Platform-specific specifications. Definitions specific to PC Client can be found in *PC Client Specific Platform TPM Profile for TPM 2.0 [6],* section Non-volatile Storage.

2.3 Privacy Protection

In TPM 2.0, privacy-sensitive operations are controlled by the Privacy Administrator. The Privacy Administrator controls the Endorsement hierarchy and sets the hierarchy authorization and policy (endorsementAuth and endorsementPolicy). The Privacy Administrator and the Owner are often the same entity.

Because an Endorsement Key is unique to a TPM and usually has a long lifetime, it could be used to identify a user or a platform. Therefore, the EK may be privacy-sensitive. The following applies if protection of privacy is important:

- The usage of the EK SHOULD be limited by its object attributes, so the EK can only be authorized per its policy (or in some cases by a password authorization, see B.4). The EK SHOULD be defined as a non-duplicable restricted decryption key. This prevents the EK from being used for signing operations.

The availability of the EK can be controlled with the flag ehEnable. The purpose of the flag is to enable and disable the Endorsement hierarchy. When the Endorsement hierarchy is disabled (ehEnable CLEAR), objects defined under that hierarchy are inaccessible, and endorsementAuth and endorsementPolicy cannot be used for authorization. It is not possible to use the EK in any command or read the public EK with TPM2_ReadPublic(). The ehEnable flag may be cleared with the command TPM2_HierarchyControl() using Endorsement Authorization or Platform Authorization.

Protection for the EK Credential can be provided by the flag phEnableNV if TPMA_NV_PLATFORMCREATE is SET in the NV Index attributes of the EK Credential. This attribute indicates whether the index was defined by the platform. When phEnableNV is CLEAR, NV space defined by the platform firmware is not accessible, including the EK Credential. This flag can only be cleared by Platform Authorization, which is seldom accessible to the end user..

3 X.509 ASN.1 Definitions

This section contains the format for the EK Credential instantiated as an X.509 certificate. All fields are defined in ASN.1 and encoded using DER [19]. The appropriate OIDs are defined in section 4.

Version 3 of the X.509 certificate structure is used for compatibility with existing PKI tools and services. TCG credential profiles do not utilize all aspects of X.509 defined fields and some fields are overloaded with TCG specific interpretations. The following sections define TCG interpretations for X.509 certificates.

TCG defines a number of new attribute value types to hold TCG-specific values. When present in a public key certificate they are carried in the subject alternative name or subject directory attributes extension.

This specification is a profile of RFC 5280 [12] which is itself a profile of the ISO/IEC/ITU-T X.509 specifications for public key certificates. All syntax and semantics are inherited from those specifications unless explicitly documented otherwise below.

3.1 TCG Attributes

3.1.1 TPM Security Assertions

These attributes describe security-related assertions about the TPM.

Each attribute begins with a version number which identifies the version of the assertion syntax. Future versions of this profile may add new assertions by appending new fields at the end of the ASN.1 SEQUENCE and increasing the version number to identify which version of the assertion syntax is encoded.

The **fieldUpgradable** BOOLEAN indicates whether the TPM is capable of having its firmware upgraded after manufacturing.

The **ekGenerationType** indicates how the Endorsement Key in the TPM was created. It may be internally generated within the TPM, generated externally and then inserted under a controlled environment during manufacturing. The revocable variants indicate whether the EK Credential can be revoked or not.

In the **CommonCriteriaMeasures**, the profile and target for the evaluation can be described by either an OID, a URI to a document describing the value, or both. If both are present, they must represent consistent values. The URI values are included in an **URIReference** which describes the URI to the document and a cryptographic hash value which identifies a specific version of the document.

URIMAX is a constant used to provide an upper bound on the length of a URI included in the certificate. This upper bound may be helpful to consumers of the extension and also helps limit the overall size of the certificate. In order to provide a reasonable upper bound for ASN.1 parsers, URIMAX SHOULD NOT exceed a value of 1024. This value was selected as it matches the length limit for <A> anchors in HTML as specified by the SGML declaration (LITLEN) for HTML [18].

STRMAX is a constant defining the upper bound on the length of a string type. Like the URIMAX this is to aid ASN.1 parsers and help limit the upper bound on the length of the certificate. Based on the expected sizes of the strings in the ASN.1 in this document an upper bound of 256 was selected. STRMAX SHOULD NOT exceed a value of 256.

```
Version ::= INTEGER { v1(0) }


tPMSecurityAssertions ATTRIBUTE ::= {
    WITH SYNTAX TPMSecurityAssertions
```

```
            ID tcg-at-tpmSecurityAssertions }

        TPMSecurityAssertions ::= SEQUENCE {
            version Version DEFAULT v1,
            fieldUpgradable BOOLEAN DEFAULT FALSE,
            ekGenerationType [0] IMPLICIT EKGenerationType OPTIONAL,
            ekGenerationLocation [1] IMPLICIT EKGenerationLocation OPTIONAL,
            ekCertificateGenerationLocation [2] IMPLICIT
                EKCertificateGenerationLocation OPTIONAL,
            ccInfo [3] IMPLICIT CommonCriteriaMeasures OPTIONAL,
            fipsLevel [4] IMPLICIT FIPSLevel OPTIONAL,
            iso9000Certified [5] IMPLICIT BOOLEAN DEFAULT FALSE,
            iso9000Uri IA5STRING (SIZE (1..URIMAX) OPTIONAL }

        EKGenerationType ::= ENUMERATED {
            internal (0),
            injected (1),
            internalRevocable(2),
            injectedRevocable(3) }

        EKGenerationLocation ::= ENUMERATED {
            tpmManufacturer (0),
            platformManufacturer (1),
            ekCertSigner (2) }

        EKCertificateGenerationLocation ::= ENUMERATED {
            tpmManufacturer (0),
            platformManufacturer (1),
            ekCertSigner (2) }

        -- common criteria evaluation

        CommonCriteriaMeasures ::= SEQUENCE {
            version IA5STRING (SIZE (1..STRMAX)), -- "2.2" or "3.1"; future syntax defined by
CC
            assurancelevel EvaluationAssuranceLevel,
            evaluationStatus EvalutionStatus,
            plus BOOLEAN DEFAULT FALSE,
            strengthOfFunction [0] IMPLICIT StrengthOfFunction OPTIONAL,
            profileOid [1] IMPLICIT OBJECT IDENTIFIER OPTIONAL,
            profileUri [2] IMPLICIT URIReference
         OPTIONAL,
            targetOid [3] IMPLICIT OBJECT IDENTIFIER OPTIONAL,
```

```
    targetUri [4] IMPLICIT URIReference OPTIONAL }

EvaluationAssuranceLevel ::= ENUMERATED {
    level1 (1),
    level2 (2),
    level3 (3),
    level4 (4),
    level5 (5),
    level6 (6),
    level7 (7) }

StrengthOfFunction ::= ENUMERATED {
    basic (0),
    medium (1),
    high (2) }

-- Reference to external document containing information relevant to this subject.
-- The hashAlgorithm and hashValue MUST both exist in each reference if either
-- appear at all.
URIReference ::= SEQUENCE {
    uniformResourceIdentifier IA5String (SIZE (1..URIMAX),
    hashAlgorithm AlgorithmIdentifier OPTIONAL,
    hashValue BIT STRING OPTIONAL }

EvaluationStatus ::= ENUMERATED {
    designedToMeet (0),
    evaluationInProgress (1),
    evaluationCompleted (2) }

-- fips evaluation

FIPSLevel ::= SEQUENCE {
    version IA5STRING (SIZE (1..STRMAX)), -- "140-1" or "140-2"
    level SecurityLevel,
    plus BOOLEAN DEFAULT FALSE }

SecurityLevel ::= ENUMERATED {
    level1 (1),
    level2 (2),
    level3 (3),
    level4 (4) }
```

3.1.2 TPM Device Attributes

The following definitions define the syntax of the relative distinguished names (RDNs) used in the subject alternative name extension to identify the type of the TPM.

The value of the **TPMManufacturer** attribute MUST be the ASCII representation of the hexadecimal value of the 4 byte vendor identifier defined in the TCG *Vendor ID Registry [3]*. Each byte is represented individually as a two digit unsigned hexadecimal number using the characters 0-9 and A-F. The result is concatenated together to form an 8 character name which is appended after the lower-case ASCII characters “id:”.

For example, the vendorId 0x12 0x34 0x56 0xEF would be encoded as “id:123456EF”.

Likewise, the value of the **TPMVersion** attribute MUST be the ASCII representation of the hexadecimal value of the 4 bytes derived from the major and minor firmware version of the TPM. The TPM firmware version is a manufacturer-specific implementation version of the TPM. The version represents the TPM firmware at the time the certificate was created, typically the initial TPM firmware loaded during manufacturing. Each byte is represented individually as a two digit unsigned hexadecimal number using the characters 0-9 and A-F. The result is concatenated together to form a 8 character name which is appended after the lower-case ASCII characters “id:”.

For example, a revMajor of 0x0002 and revMinor of 0x0008 would be encoded as “id:00020008”.

The value of the **TPMModel** attribute is a UTF 8 string that represents the TPM part number. The values are manufacturer-specific.

```
TPMManufacturer ATTRIBUTE ::= {
    WITH SYNTAX UTF8String (SIZE (1..STRMAX))
    ID tcg-at-tpmManufacturer }

TPMModel ATTRIBUTE ::= {
    WITH SYNTAX UTF8String (SIZE (1..STRMAX))
    ID tcg-at-tpmModel }

TPMVersion ATTRIBUTE ::= {
    WITH SYNTAX UTF8String (SIZE (1..STRMAX))
    ID tcg-at-tpmVersion }
```

3.1.3 TPM Specification Attributes

The following definitions define the syntax of the TPM specification attributes.

The **TPMSpecification** attribute identifies the TPM family, level and revision of the TPM specification with which a TPM implementation is compliant. The family value of “2.0” with level 0 and revision 99 identifies a TPM compliant with a public TPM 2.0 specification version 0.99 published by TCG. The family value is encoded in a UTF 8 string but the current defined standard values fall within the ASCII character set.

```
tPMSpecification ATTRIBUTE ::= {
    WITH SYNTAX TPMSpecification
    ID tcg-at-tpmSpecification }

TPMSpecification ::= SEQUENCE {
    family UTF8String (SIZE (1..STRMAX)),
    level INTEGER,
    revision INTEGER }
```

3.2 EK Certificate

This section contains the format for a TPM 2.0 EK Credential conforming to this specification. An X.509 EK certificate is an instantiation of the TPM EK Credential defined in section 2.2.

The “Field Status” column in the table below specifies the presence of the certificate fields. The value “Standard” means the field is an inherent component of the standard certificate syntax and is not optional. The value “MUST”, “SHOULD” or “MAY” is used to indicate the presence of the certificate extensions. The content is described in the “Value” column. Values marked with “(optional)” are added for completeness and are meant to be optional.

NOTE This specification does not preclude the use of other certificate extensions. However, any extensions marked as critical will cause interoperability problems when existing clients do not know how to parse the extension and reject it as specified in RFC 5280 [12], section 4.2. (This has historically been a challenge when introducing new critical extensions.)

Field Name	**RFC 5280 Type**	**Value**	**Field Status**
Version	INTEGER	V3 (encoded as value 2)	Standard
Serial Number	INTEGER	Positive integer	Standard
Signature Algorithm	AlgorithmIdentifier	algorithm-specific, see C.1	Standard
Issuer	Name	Name of issuing CA	Standard
Validity	notBefore notAfter	Beginning and end of validity period	Standard
Subject	Name	Unique name assigned by the manufacturer or empty	Standard
Subject Public Key Info	SubjectPublicKeyInfo	algorithm-specific, see C.2	Standard
Extensions			
Certificate Policies	CertificatePolicies	PolicyIdentifier	MAY non-critical
Subject Alternative Name	GeneralName directoryName	TPMManufacturer TPMModel TPMVersion	MUST critical/ non-critical (dep. on subject)
Basic Constraints	BasicConstraints	CA=FALSE	MUST critical
Subject Directory Attributes	SubjectDirectoryAttributes	TPMSpecification Family Level Revision	MAY non-critical

Field Name	RFC 5280 Type	Value	Field Status
Authority Key Identifier	AuthorityKeyIdentifier	keyIdentifier (must) authorityCertIssuer (optional) authorityCertSerialNumber (optional)	MUST non-critical
Authority Info Access	AuthorityInfoAccessSyntax	id-ad-caIssuers URI to issuing CA id-ad-ocsp URI to OCSP responder	MAY non-critical
CRL Distribution	CRLDistributionPoints	URI to CRL	MAY non-critical
Key Usage	KeyUsage	keyEncipherment (RSA EK) or keyAgreement (ECC EK) or digitalSignature	MUST critical
Extended Key Usage	ExtKeyUsageSyntax	tcg-kp-EKCertificate	MAY non-critical
Subject Key Id	SubjectKeyIdentifier	Key identifier	MAY

Table 1: EK Certificate Fields

3.2.1 Version

This field describes the version of the X.509 certificate. Since EK certificates contain mandatory extensions the version number MUST be set to 3 (which is encoded as the value 2 in ASN.1).

3.2.2 Serial Number

The serial number MUST be a positive integer which is uniquely assigned to each EK certificate by the issuer. The combination of an issuer's DN and the serial number MUST uniquely describe a single certificate.

3.2.3 Signature Algorithm

This field identifies the algorithm used by the EK certificate issuer to sign the certificate. This field is algorithm-specific, see annex C.1.

3.2.4 Issuer

This field contains the distinguished name of the certificate issuer which is the entity that vouches that the TPM is genuine and complies with the TPM 2.0 Library Specification [1].

3.2.5 Validity

The period when the certificate is valid is represented by two date values named notBefore and notAfter. Issuers SHOULD assign notBefore to the current time when the EK certificate is issued and notAfter to the last date that the certificate will be considered valid. Both notBefore and notAfter MUST use the appropriate time format as indicated by RFC 5280 [12]. (See also section 2.2.2 EK Credential Lifetime)

3.2.6 Subject

The subject field MUST contain an X.500 distinguished name (DN) that uniquely identifies the TPM or, if unique identification through the subject field is not required, MUST be empty.

If the subject name field is empty, the subject alternative name extension MUST be critical in accordance with RFC 5280 [12], otherwise it SHOULD be non-critical.

The subject field MAY contain a device (TPM) serial number in the attribute id-at-serialNumber.

NOTE The TPM serial number usually includes detailed production parameters. Since that might reveal information that the manufacturer does not want to disclose, the hash of the TPM serial number could be used instead in the EK Credential.

3.2.7 Subject Public Key Info

This describes the public Endorsement Key algorithm and key value. This field is algorithm-specific, see annex C.2.

3.2.8 Certificate Policies

This extension indicates the policy terms under which the certificate was issued. This extension is optional. If included, it SHOULD be non-critical and PolicyIdentifier MUST have at least one object identifier. Policy qualifiers, such as the cPSuri policy qualifier and the userNotice policy qualifier SHOULD NOT be included.

NOTE A pointer to a Certification Practice Statement (CPS) can be provided in the manufacturer's datasheet instead.

3.2.9 Subject Alternative Name

This contains the alternative name of the entity associated with this certificate. The issuer MUST include TPM manufacturer, TPM part number and TPM firmware version, using the directoryName-form within the GeneralName structure. The ASN.1 encoding is specified in section 3.1.2 TPM Device Attributes. In accordance with RFC 5280 [12], this extension MUST be critical if subject is empty and SHOULD be non-critical if subject is non-empty.

- The TPM manufacturer identifies the manufacturer of the TPM. This value MUST be the vendor ID defined in the TCG *Vendor ID Registry [3]* . It MUST match the value reported by the command TPM2_GetCapability(property = TPM_PT_MANUFACTURER).
- The TPM part number is encoded as a string and is manufacturer-specific. A manufacturer MUST provide a way to the user to retrieve the part number physically or logically. This information could be e.g. provided as part of the vendor string in the command TPM2_GetCapability(property = TPM_PT_VENDOR_STRING_x; x=1…4).
- The TPM firmware version is a manufacturer-specific implementation version of the TPM. The version represents the TPM firmware at the time the certificate was created, typically the initial TPM firmware loaded during manufacturing. In the case of a Field Upgrade, the firmware version will change and this value will no longer match the version reported by the command TPM2_GetCapability (property = TPM_PT_FIRMWARE_VERSION_1).

Version 2.0 of this specification [5] allowed the inclusion of an optional attribute, called HardwareModuleName that contains the TPM serial number. HardwareModuleName SHOULD NOT be used in EK certificates conforming to this version of this specification. Instead, if a serial number is present, it SHOULD be stored in the subject field (see 3.2.6).

3.2.10 Basic Constraints

This extension indicates whether the subject is a CA. "CA" MUST be set to FALSE. This extension MUST be critical.

3.2.11 Subject Directory Attributes

The extension includes miscellaneous properties and security assertions about the entity. This extension MUST be non-critical.

The following attribute MAY be included in a subject directory attributes extension in the EK certificate:

- The “TPM Specification” attribute that identifies the family, level and revision of the TCG TPM specification to which the TPM was designed. The ASN.1 encoding is specified in section 3.1.3 TPM Specification Attributes.

Version 2.0 of this specification [5] allowed the inclusion of an optional attribute, called “TPM Security Assertions”, which described various assertions about the security properties of the TPM and the conditions under which the Endorsement Key was generated. The Security Assertions attribute SHOULD NOT be included in EK certificates conforming to this version of this specification.

3.2.12 Authority Key Identifier

This identifies the subject public key of the certificate issuer and hence facilitates the validation of the certificate path. The certificate MUST contain an AuthorityKeyIdentifier that matches the subject key identifier of the CA certificate. The issuer name and the serial number are optional. This extension MUST be non-critical.

3.2.13 Authority Information Access

This extension provides additional information about the issuer. Authority Information Access MAY contain the accessMethod OID id-ad-caIssuers (see note 1 below) and/ or the OID id-ad-ocsp. This extension MUST be non-critical.

If id-ad-caIssuers appears as accessMethod, then the accessLocation value SHOULD point to the URL where the certificate of the issuing CA can be retrieved.

NOTE 1 For TPMs designed to meet Windows [22], the EK certificate MUST contain an AIA extension that contains the URL for the issuing CA certificate in the certificate chain, and the AIA extension MUST also be present in each non-root certificate in the chain.

NOTE 2 The root certificate should not be retrieved from a URL in the AIA extension as this is a possible attack vector. Instead, the root certificate should be provisioned in a trusted root store out of band or from a trusted source. Alternatively, a hash of the root public key or certificate could be stored in a device root of trust.

If id-ac-ocsp appears as accessMethod, then the accessLocation value SHOULD point to the access value of the OCSP responder (HTTP URI). The relying party can access the certificate status for this certificate by sending a properly formatted OCSPRequest to the URI. If both a CDP and OCSP AIA extension are present in the certificate, then the relying parties SHOULD use OCSP as the primary validation mechanism.

3.2.14 CRL Distribution

This extension is optional and provides the location of the subject’s revocation information. The relying party can access the CRL for this certificate from this URI. If both a CDP and OCSP AIA extension are present in the certificate, then relying parties SHOULD use OCSP as the primary validation mechanism. This extension MUST be non-critical.

3.2.15 Key Usage

This extension indicates the intended purpose of the subject public key. This extension MUST be critical. The usage of the Endorsement Key is defined by the object attributes and algorithm in its public area template.

If the EK has the object attributes

- *restricted*, *decrypt* SET (i.e. the EK is a Storage key),
    - the keyEncipherment bit MUST be set for an RSA EK certificate
    - the keyAgreement bit MUST be set for an ECC EK certificate

- *restricted, sign* SET (i.e. the EK is a Signing key), the digitalSignature bit MUST be set

3.2.16 Extended Key Usage

The extended key usage extension indicates the intended purpose of the subject public key and MAY be included in the EK certificate (see note 1 below). If present, extended key usage SHOULD contain the OID tcg-kp-EKCertificate defined in section 5 of this document as shown below. The OID is used to unambiguously identify the certificate as an EK certificate. This extension MUST be non-critical.

```
tcg-kp-EKCertificate OBJECT IDENTIFIER ::= {
    joint-iso-itu-t(2) international-organizations(23) tcg(133) kp(8) 1}
```

NOTE 1 For TPMs designed to meet Windows [22], the extended key usage MUST contain the OID tcg-kp-EKCertificate.

NOTE 2 If the issuing CA is used exclusively to issue EK certificates, the OID tcg-kp-EKCertificate MAY also be included in the issuing CA certificate. This ensures that the use of the CA is limited to that particular purpose. If the issuing CA issues certificates for multiple known purposes, then the set of relevant EKU OIDs could be included in the issuing CA certificate.

3.2.17 Subject Key Identifier

This identifies the public key of the certificate. This extension MAY be included. If included, it MUST be non-critical.

4 X.509 ASN.1 Structures and OIDs

TCG has registered an object identifier (OID) namespace as an “international body” in the ISO registration hierarchy. This leads to shorter OIDs and gives TCG the ability to manage its own namespace. The OID namespace is inherited from TCPA. These definitions are intended to be used within the context of an X.509 v3 certificate specifically leveraging the profile described in RFC 5280 [12].

```
-- TCG specific OIDs
tcg OBJECT IDENTIFIER ::= {
    joint-iso-itu-t(2) international-organizations(23) tcg(133) }

tcg-attribute OBJECT IDENTIFIER ::= {tcg 2}
tcg-kp OBJECT IDENTIFIER ::= {tcg 8}

-- TCG Attribute OIDs
tcg-at-tpmManufacturer OBJECT IDENTIFIER ::= {tcg-attribute 1}
tcg-at-tpmModel OBJECT IDENTIFIER ::= {tcg-attribute 2}
tcg-at-tpmVersion OBJECT IDENTIFIER ::= {tcg-attribute 3}
tcg-at-tpmSpecification OBJECT IDENTIFIER ::= {tcg-attribute 16}
tcg-at-tpmSecurityAssertions OBJECT IDENTIFIER ::= {tcg-attribute 18}

-- TCG Key Purposes OIDs
tcg-kp-EKCertificate OBJECT IDENTIFIER ::= {tcg-kp 1}

-- tcg specification attributes for tpm

tPMSpecification ATTRIBUTE ::= {
    WITH SYNTAX TPMSpecification
    ID tcg-at-tpmSpecification }

TPMSpecification ::= SEQUENCE {
    family UTF8String (SIZE (1..STRMAX)),
    level INTEGER,
    revision INTEGER }

-- manufacturer implementation model and version attributes

TPMManufacturer ATTRIBUTE ::= {
    WITH SYNTAX UTF8String (SIZE (1..STRMAX))
    ID tcg-at-tpmManufacturer }

TPMModel ATTRIBUTE ::= {
    WITH SYNTAX UTF8String (SIZE (1..STRMAX))
    ID tcg-at-tpmModel }

TPMVersion ATTRIBUTE ::= {
    WITH SYNTAX UTF8String (SIZE (1..STRMAX))
    ID tcg-at-tpmVersion }

-- tpm security assertions

Version ::= INTEGER { v1(0) }

tPMSecurityAssertions ATTRIBUTE ::= {
    WITH SYNTAX TPMSecurityAssertions
    ID tcg—at-tpmSecurityAssertions
}

TPMSecurityAssertions ::= SEQUENCE {
    version Version DEFAULT v1,
    fieldUpgradable BOOLEAN DEFAULT FALSE,
    ekGenerationType [0] IMPLICIT EKGenerationType OPTIONAL,
```

```
            ekGenerationLocation [1] IMPLICIT EKGenerationLocation OPTIONAL,
            ekCertificateGenerationLocation [2] IMPLICIT
                EKCertificateGenerationLocation OPTIONAL,
            ccInfo [3] IMPLICIT CommonCriteriaMeasures OPTIONAL,
            fipsLevel [4] IMPLICIT FIPSLevel OPTIONAL,
            iso9000Certified [5] IMPLICIT BOOLEAN DEFAULT FALSE,
            iso9000Uri IA5STRING (SIZE (1..URIMAX)) OPTIONAL }


        EKGenerationType ::= ENUMERATED {
            internal (0),
            injected (1),
            internalRevocable(2),
            injectedRevocable(3) }


        EKGenerationLocation ::= ENUMERATED {
            tpmManufacturer (0),
            platformManufacturer (1),
            ekCertSigner (2) }


        EKCertificateGenerationLocation ::= ENUMERATED {
            tpmManufacturer (0),
            platformManufacturer (1),
            ekCertSigner (2) }


        -- common criteria evaluation

        CommonCriteriaMeasures ::= SEQUENCE {
            version IA5STRING (SIZE (1..STRMAX)), -- "2.2" or "3.1"; future syntax defined by
CC
            assurancelevel EvaluationAssuranceLevel,
            evaluationStatus EvalutionStatus,
            plus BOOLEAN DEFAULT FALSE,
            strengthOfFunction [0] IMPLICIT StrengthOfFunction OPTIONAL,
            profileOid [1] IMPLICIT OBJECT IDENTIFIER OPTIONAL,
            profileUri [2] IMPLICIT URIReference OPTIONAL,
            targetOid [3] IMPLICIT OBJECT IDENTIFIER OPTIONAL,
            targetUri [4] IMPLICIT URIReference OPTIONAL }


        EvaluationAssuranceLevel ::= ENUMERATED {
            level1 (1),
            level2 (2),
            level3 (3),
            level4 (4),
            level5 (5),
            level6 (6),
            level7 (7) }

        StrengthOfFunction ::= ENUMERATED {
            basic (0),
            medium (1),
            high (2) }

        URIReference ::= SEQUENCE {
            uniformResourceIdentifier IA5String (SIZE (1..URIMAX)),
            hashAlgorithm AlgorithmIdentifier OPTIONAL,
            hashValue BIT STRING OPTIONAL }


        EvaluationStatus ::= ENUMERATED {
            designedToMeet (0),
            evaluationInProgress (1),
            evaluationCompleted (2) }


        -- fips evaluation

        FIPSLevel ::= SEQUENCE {
            version IA5STRING (SIZE (1..STRMAX)), -- "140-1" or "140-2"
            level SecurityLevel,
```

```
        plus BOOLEAN DEFAULT FALSE }

    SecurityLevel ::= ENUMERATED {
        level1 (1),
        level2 (2),
        level3 (3),
        level4 (4) }
```

5 References

For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

[1] TPM 2.0 Library Specification:
http://www.trustedcomputinggroup.org/resources/tpm_library_specification

[2] Registry of Reserved TPM 2.0 Handles and Localities:
http://www.trustedcomputinggroup.org/resources/registry

[3] Vendor ID Registry:
http://www.trustedcomputinggroup.org/resources/vendor_id_registry

[4] TCG Credential Profile Version 1.0, Version 1.1 and Version 1.2:
http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_tcg_credential_profiles_specification

[5] TCG EK Credential Profile for TPM 2.0, Version 2.0, Revision 14:
https://trustedcomputinggroup.org/tcg-ek-credential-profile-tpm-family-2-0

[6] PC Client Specific Platform TPM Profile for TPM 2.0:
http://www.trustedcomputinggroup.org/developers/pc_client

[7] CMC Profile for EK/Platform Certificate Enrollment for TPM 1.2:
http://www.trustedcomputinggroup.org/developers/infrastructure

[8] TPM Keys for Platform Identity for TPM 1.2:
http://www.trustedcomputinggroup.org/developers/infrastructure

[9] TPM 1.2 Main Specification:
http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[10] IEEE Standard for Local and metropolitan area networks, Secure Device Identity, 2009

[11] NIST Special Publication 800-57, Recommendation for Key Management – Part 1: General

[12] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 5280, http://www.ietf.org/rfc/rfc5280.txt

[13] Using SHA2 Algorithms with Cryptographic Message Syntax, RFC 5754,
http://www.ietf.org/rfc/rfc5754.txt

[14] Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 3279, http://www.ietf.org/rfc/rfc3279.txt

[15] Elliptic Curve Cryptography Subject Public Key Information, RFC 5480,
http://www.ietf.org/rfc/rfc5480.txt

[16] Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages, RFC 4108,
http://www.ietf.org/rfc/rfc4108.txt

[17] Key words for use in RFCs to Indicate Requirement Levels, RFC 2119,
http://www.ietf.org/rfc/rfc2119.txt

[18] Hypertext Markup Language – 2.0, RFC 1866, http://www.ietf.org/rfc/rfc1866.txt

[19] ITU-T X.690: Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

[20] Cryptographic Application Identifier Criterion Specification, GM/T 0006-2012,
http://www.cssn.net.cn/pagesnew/search/standard_detail.jsp?a001=NjM0ODU5Mg==

[21] TCG Glossary: https://trustedcomputinggroup.org/glossary

[22] Hardware Compatibility Specification for Systems for Windows 10

[23] CNSA-Suite-and-Quantum-Computing-FAQ.pdf, https://www.nsa.gov/

A. Certificate Examples

A.1 Example 1 (user device TPM, e.g. PC-Client)

This annex provides an example for a standard, user device TPM (e.g. PC-Client) Endorsement Key certificate. The ASN.1 encoding for the subject alternative name and subject directory attributes extension is provided below. The values used in this example are for illustrative purposes and must be replaced with manufacturer-specific data.

Subject alternative name:
TPMManufacturer = id:54534700 (TCG)
TPMModel = ABCDEF123456 (part number)
TPMVersion = id:00010023 (firmware version)

```
// SEQUENCE
30 49
    // SET
    31 16
        // SEQUENCE
        30 14
            // OBJECT IDENTIFER tcg-at-tpmManufacturer (2.23.133.2.1)
            06 05 67 81 05 02 01
            // UTF8 STRING id:54434700 (TCG)
            0C 0B 69 64 3A 35 34 34 33 34 37 30 30
    // SET
    31 17
        // SEQUENCE
        30 15
            // OBJECT IDENTIFER tcg-at-tpmModel (2.23.133.2.2)
            06 05 67 81 05 02 02
            // UTF8 STRING ABCDEF123456
            0C 0C 41 42 43 44 45 46 31 32 33 34 35 36
    // SET
    31 16
        // SEQUENCE
        30 14
            // OBJECT IDENTIFER tcg-at-tpmVersion (2.23.133.2.3)
            06 05 67 81 05 02 03
            // UTF8 STRING id:00010023
            0C 0B 69 64 3A 30 30 30 31 30 30 32 33
```

Subject directory attributes:
TPMSpecification
Family = id:322E3000 (2.0)
Level = 0
Revision = 99
TPMSecurityAssertions (not included here since optional)

```
// SEQUENCE
30 16
    // OBJECT IDENTIFIER tcg-at-tpmSpecification (2.23.133.2.16)
    06 05 67 81 05 02 10
        // SET
        31 0D
            // SEQUENCE
            30 0B
                // UTF8 STRING (2.0)
                0C 03 32 2E 30
                // INTEGER (0)
                02 01 00
                // INTEGER (99)
                02 01 63
```

The encoding of the extensions above is extracted from the following example certificate. The example certificate provided below is for illustrative proposes only, and all example values must be replaced with manufacturer-specific data. For simplicity some optional configurations (e.g. optional data within an extension) are omitted. The manufacturer's certificate is not required to look exactly the same as this example certificate. For easier testing the certificate is provided in PEM format. When read from the TPM the certificate is encoded in DER [19].

```
-----BEGIN CERTIFICATE-----
MIID7zCCAtegAwIBAgIBATANBgkqhkiG9w0BAQsFADAUMRIwEAYDVQQDDAlFeGFt
cGxlQ0EwHhcNMTQwMTE1MTU0MDUwWhcNMTUwMTE1MTU0MDUwWjAAMIIBIjANBgkq
hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAncvm0aOBK05rdNInYXzJGV5SFteVUFpt
XFxg4evROvlulB3BzUmFGQYFDcItVnJX2fAvf0UJLtLBVBQggb5ylL6bRpj72cS3
oyNbs0CGmix9Z1QDjkZZFvIsD1GcKO0tvsCvsEItH8Cm0fq8WcGFijWLdRD5eulP
55pq1bAHAvIo4+VLMJVBG71xrKGZeHPjKoq6seYjh7AGy+hk2vmFzpZ8Ghdgqv+K
02IZ7FEdzuylHW8U3qsxBHysMut4inj6AiVf467OOs5meHiifIK9MGkovMrfY9iX
uUVUs/KXpE1sgeoX9BLvx1BPcODosr5K+z5i71OtIXy4CXrPvcGzRwIDAQABo4IB
XjCCAVowQAYIKwYBBQUHAQEENDAyMDAGCCsGAQUFBzAChiRodHRwOi8vd3d3LmV4
YW1wbGUuY29tL0V4YW1wbGVDQS5jcnQwDgYDVR0PAQH/BAQDAgAgMFkGA1UdEQEB
/wRPME2kSzBJMRYwFAYFZ4EFAgEMC2lkOjU0NDM0NzAwMRcwFQYFZ4EFAgIMDEFC
Q0RFRjEyMzQ1NjEWMBQGBWeBBQIDDAtpZDowMDAxMDAyMzAMBgNVHRMBAf8EAjAA
MDUGA1UdHwQuMCwwKqAooCaGJGh0dHA6Ly93d3cuZXhhbXBsZS5jb20vRXhhbXBs
ZUNBLmNybDAQBgNVHSAECTAHMAUGAyoDBDAfBgNVHSMEGDAWgBQ0d2ckTESv554q
4LJMaVeVJLM92jAQBgNVHSUECTAHBgVngQUIATAhBgNVHQkEGjAYMBYGBWeBBQIQ
MQ0wCwwDMi4wAgEAAgFjMA0GCSqGSIb3DQEBCwUAA4IBAQAba2btJ/+4z02MWpNp
99AFGpEu3yIaJqI6NeHvC6fxxe/9lWlHKISR+CnpAh03/MKT8TP2/cUSi0jjkQNh
MtueUNofE79fYXtHXHU7wzzUFWNwCmhTuHDYl3jmD0fJ9yA2CuUHT6q3UV+PwXN+
EHE1hQwC8QtNC/5A7wY1e5dBLdgwSSIgTc4lSsbNcZ9d+m7mWEWpumSYU0czTDEN
Hmdu/VJuDN/RCOAyBb+hc19LAucGmnFYOhxWHfd9zbXZA1ldFUxrpPuVfKx+Eo8f
rMsB2oZKMwSYUAWotqolhLe2wdBMRjdmVz44kIhuFB7y4BpQjlB1+xAzX9Hb31CG
eoS2
-----END CERTIFICATE-----
```

B. Default EK Templates (algorithm-specific)

B.1 Introduction

This annex defines multiple default EK Templates for algorithm-specific Endorsement Keys. The Templates are assigned to either the Low or High Range. These default values are used to generate the RSA or ECC Endorsement Key corresponding to the EK certificates installed by the TPM or Platform manufacturer. The hash calculated over the public area template is one of the command parameters that is used to create the EK.

The TPM or Platform manufacturer MAY create a proprietary EK public area template that is different from the defaults in this annex (this procedure is further described in section 2.2.1). Platform-specific working groups MAY define a proprietary EK public area template if required. The EK provided by the manufacturer MUST be defined as a non-duplicable key. This is ensured by setting the attributes *fixedTPM* and *fixedParent* both to SET (1). The noDA attribute is set to CLEAR (0) (meaning the object is subject to dictionary attack protections), in the default EK Templates because the EK is considered privacy sensitive. When privacy is an issue, great care should be taken when selecting the attributes and the authorization for the key (see section 2.3 privacy protection).

B.2 Backwards Compatibility

Version 2.0 of the EK Credential Profile [5] defined one EK Template for RSA 2048 and one for ECC NIST P256. These Templates are retained unmodified in this version of the EK Credential Profile and are named L-1 (see B.3.3) and L-2 (see B.3.4). In addition, a second EK Template for RSA 2048 and ECC NIST P256 are defined, which utilize a different authorization.

If a platform-specific TPM profile requires an EK Certificate from the manufacturer for RSA 2048 and/ or ECC NIST P256, the platform specification SHOULD specify which EK Template is to be used to create the associated EK. For backwards compatibility, it is recommended that TPMs use the Templates L-1 and L-2 in the Low Range instead of H-1 and H-2 in the High Range.

B.3 EK Templates in the Low Range

B.3.1 Introduction

The EK certificates associated with the default EK Templates defined in this section are stored in the Low Range (see section 2.2.1.4). The default EK Templates L-1 and L-2 (see appendices B.3.3 and B.3.4) specify an RSA 2048 bit and ECC NIST P256 bit restricted decryption (Storage) key whose authorization is only allowed with *authPolicy*. The policies in Template L-1 and L-2 require *endorsementAuth* to authorize use of the EK. This policy is created with TPM2_PolicySecret() where the *authHandle* parameter of the command, indicating the entity providing the authorization value, references the Endorsement hierarchy. In the following, this policy is referenced as *PolicyA*. A symmetric key is used to protect the child keys of the EK and is defined as an AES 128 bit key using CFB mode. SHA256 is used to calculate the Name of the EK. The buffer reserved for the public key of the EK is set to all zeros. For the setting of the sensitive data, see section 2.1.1.

Using *PolicyA* rather than a policy using the EK authorization (*userWithAuth* SET) enables the following use case: An entity wishes to pre-provision an EK in persistent storage; since the password of a persistent object cannot be changed, the entity must:

- communicate the password to the final user, and
- securely delete the copy of the password held by the entity

This policy (*PolicyA*) used by the EK Templates in the Low Range permits the EK to be pre-provisioned while letting the end user set its password (though endorsement authorization).

The disadvantage of *PolicyA* is that it results in reduced flexibility for the Privacy Administrator when delegating control of the EK. If an administrator wishes to give an end user the ability to use the EK, the administrator has to give the end user the Endorsement hierarchy password. Among other things,

knowledge of the EH password allows the end user to change the EH password and policy, potentially locking out the administrator from using the EK.

B.3.2 Satisfying *PolicyA*

When using the EK created with Template L-1 or L-2, the user has to satisfy *PolicyA*. This is done by executing the TPM2_PolicySecret() command, passing in the handle to the Endorsement Hierarchy, and then proving knowledge of the Endorsement Hierarchy password. The caller proves knowledge of the Endorsement Hierarchy password using an authorization session. A password session, an HMAC session, or a policy session containing TPM2_PolicyAuthValue() or TPM2_PolicyPassword() will satisfy this requirement.

B.3.3 Template L-1: RSA 2048 (Storage)

Parameter	Type	Content
type	TPMI_ALG_PUBLIC	TPM_ALG_RSA
nameAlg	TPMI_ALG_HASH	TPM_ALG_SHA256
objectAttributes	TPMA_OBJECT	fixedTPM = 1 stClear = 0 fixedParent = 1 sensitiveDataOrigin = 1 userWithAuth = 0 adminWithPolicy = 1 noDA = 0 encryptedDuplication = 0 restricted = 1 decrypt = 1 sign = 0
authPolicy	TPM2B_DIGEST	
size	UINT16	32
buffer	BYTE	0x83, 0x71, 0x97, 0x67, 0x44, 0x84, 0xB3, 0xF8, 0x1A, 0x90, 0xCC, 0x8D, 0x46, 0xA5, 0xD7, 0x24, 0xFD, 0x52, 0xD7, 0x6E, 0x06, 0x52, 0x0B, 0x64, 0xF2, 0xA1, 0xDA, 0x1B, 0x33, 0x14, 0x69, 0xAA TPM2_PolicySecret(TPM_RH_ENDORSEMENT)
parameters	TPMS_RSA_PARMS	
symmetric->algorithm	TPMI_ALG_SYM_OBJECT	TPM_ALG_AES
symmetric->keyBits	TPMI_AES_KEY_BITS	128
symmetric->mode	TPMI_SYM_MODE	TPM_ALG_CFB
symmetric->details		NULL
scheme->scheme	TPMI_ALG_ASYM_SCHEME	TPM_ALG_NULL
scheme->details		NULL
keyBits	TPMI_RSA_KEY_BITS	2048
exponent	UINT32	0
unique	TPM2B_PUBLIC_KEY_RSA	
size	UINT16	256
buffer	BYTE	All 0

Table 2: Default EK Template (TPMT_PUBLIC) L-1: RSA 2048 (Storage)

B.3.4 Template L-2: ECC NIST P256 (Storage)

Parameter	Type	Content
type	TPMI_ALG_PUBLIC	TPM_ALG_ECC
nameAlg	TPMI_ALG_HASH	TPM_ALG_SHA256
objectAttributes	TPMA_OBJECT	fixedTPM = 1 stClear = 0 fixedParent = 1 sensitiveDataOrigin = 1 userWithAuth = 0 adminWithPolicy = 1 noDA = 0 encryptedDuplication = 0 restricted = 1 decrypt = 1 sign = 0
authPolicy	TPM2B_DIGEST	
size	UINT16	32
buffer	BYTE	0x83, 0x71, 0x97, 0x67, 0x44, 0x84, 0xB3, 0xF8, 0x1A, 0x90, 0xCC, 0x8D, 0x46, 0xA5, 0xD7, 0x24, 0xFD, 0x52, 0xD7, 0x6E, 0x06, 0x52, 0x0B, 0x64, 0xF2, 0xA1, 0xDA, 0x1B, 0x33, 0x14, 0x69, 0xAA TPM2_PolicySecret(TPM_RH_ENDORSEMENT)
parameters	TPMS_ECC_PARMS	
symmetric->algorithm	TPMI_ALG_SYM_OBJECT	TPM_ALG_AES
symmetric->keyBits	TPMI_AES_KEY_BITS	128
symmetric->mode	TPMI_SYM_MODE	TPM_ALG_CFB
symmetric->details		NULL
scheme->scheme	TPMI_ALG_ECC_SCHEME	TPM_ALG_NULL
scheme->details		NULL
curveID	TPMI_ECC_CURVE	TPM_ECC_NIST_P256
kdf->scheme	TPMI_ALG_KDF	TPM_ALG_NULL
kdf->details		NULL
unique	TPMS_ECC_POINT	
x->size	UINT16	32
x->buffer	BYTE	All 0
y->size	UINT16	32
y->buffer	BYTE	All 0

Table 3: Default EK Template (TPMT_PUBLIC) L-2: ECC NIST P256 (Storage)

B.4 EK Templates in the High Range

B.4.1 Introduction

The EK certificates associated with the default EK Templates defined in this section are stored in the High Range (see section 2.2.1.5). The default EK Templates H-1 to H-5 (see appendices B.4.4 to B.4.8) specify an RSA 2048 bit, ECC NIST P256, NIST P384, NIST P521, and SM2 P256 bit restricted decryption (Storage) key whose authorization is allowed with either the object's *authValue* (as *userWithAuth* is SET) or the object's *authPolicy.* The *authPolicy* in Template H-1 to H-5 is set to *PolicyB*.

PolicyB is a policy OR term of

- *PolicyA*: a policy created with TPM2_PolicySecret() where the entity providing the authorization value is the Endorsement hierarchy, and
- PolicyC: a policy created withTPM2_PolicyAuthorizeNV() where the authorization value is determined by the data (payload) area of a reserved NV Index.

A symmetric key is used to protect the child keys of the EK and is defined as an AES 128 bit, AES 256 bit, or SM4 128 bit key using CFB mode. The hash algorithm used to calculate the Name of the EK is defined as SHA256, SHA384, SHA512, or SM3_256 and has a security strength equivalent to the Endorsement Key. The *unique* field (reserved for the public key of the EK) is set to Empty Buffer for the RSA default EK Template, and set to Empty Point for the ECC default EK Templates. For the setting of the sensitive data, see section 2.1.1.

B.4.2 Authorization Options

The authorization of the default EK Templates defined in the High Range (H-1 to H-5) is improved in two ways (compared to the Low Range):

1) The policy of the EK is changed to an OR policy that allows authorization of the EK with
   a. Either the EH password
   b. OR a policy (only writeable with knowledge of the EH password) at a particular NV index (see B.5).
2) *userWithAuth* is SET (1) so that the EK can also be used by someone with knowledge of the EK's *authValue*.

The policy in the default high-range EK Templates allows only someone who knows the EH password (if he or she wishes) to assign (or change) a policy to use the EK. If that is wished, an NV index (at a reserved index) must be created and written with that policy. Otherwise no additional NV indexes are used.

The problem of someone creating an EK with a known *authValue* and storing it persistently in the TPM before the EH password is assigned using TPM2_HierarchyChangeAuth() can be obviated in a few ways by either

1) Clearing the TPM using TPM2_Clear() and then changing the EH password when it is recreated -OR-
2) Changing the EH password and then checking with the TPM to see if any keys are persistently stored and evicting them.

After either of these operations, only someone with EH authority is able to create a new EK, assigning it a new password.

B.4.3 Satisfying *PolicyB*

When using the EK created with one of the Templates H-1 to H-5, the user has to either prove knowledge of the object's *authValue* or satisfy *PolicyB*. *PolicyB* can be satisfied by either satisfying *PolicyA* (see B.3.2) OR by satisfying *PolicyC*. *PolicyC* is stored in a reserved NV index (defined in

B.5). The NV index is established with a policy that only allows it to be written by the Endorsement Hierarchy owner. The Name of this NV index (after it is written by the Endorsement Hierarchy owner), is used in creation of *PolicyC*.

When the TPM is shipped, the NV Index will not be *Populated* as described in B.5.1. It is entirely up to the owner of the TPM if he or she wishes to populate it. If it is not *Populated*, the EK is controlled with the Endorsement Hierarchy authorization. If the owner decides to populate the NV Index with an owner chosen policy, the EK can be controlled either with that policy OR with the Endorsement Hierarchy authorization. In this case, the owner can now delegate control of the EK via use of *PolicyB*. *PolicyB* can be satisfied either by first executing *PolicyA* and then executing TPM2_PolicyOR(). It can also be satisfied by first satisfying whatever policy was placed in the reserved NV Index, then executing TPM2_PolicyAuthorizeNV(), and then executing TPM2_PolicyOR().

B.4.4 Template H-1: RSA 2048 (Storage)

Parameter	Type	Content
type	TPMI_ALG_PUBLIC	TPM_ALG_RSA
nameAlg	TPMI_ALG_HASH	TPM_ALG_SHA256
objectAttributes	TPMA_OBJECT	fixedTPM = 1 stClear = 0 fixedParent = 1 sensitiveDataOrigin = 1 userWithAuth = 1 adminWithPolicy = 1 noDA = 0 encryptedDuplication = 0 restricted = 1 decrypt = 1 sign = 0
authPolicy	TPM2B_DIGEST	
size	UINT16	32
buffer	BYTE	0xCA, 0x3D, 0x0A, 0x99, 0xA2, 0xB9, 0x39, 0x06, 0xF7, 0xA3, 0x34, 0x24, 0x14, 0xEF, 0xCF, 0xB3, 0xA3, 0x85, 0xD4, 0x4C, 0xD1, 0xFD, 0x45, 0x90, 0x89, 0xD1, 0x9B, 0x50, 0x71, 0xC0, 0xB7, 0xA0 ($PolicyB_{SHA256}$, see B.6.2)
parameters	TPMS_RSA_PARMS	
symmetric->algorithm	TPMI_ALG_SYM_OBJECT	TPM_ALG_AES
symmetric->keyBits	TPMI_AES_KEY_BITS	128
symmetric->mode	TPMI_SYM_MODE	TPM_ALG_CFB
symmetric->details		NULL
scheme->scheme	TPMI_ALG_ASYM_SCHEME	TPM_ALG_NULL
scheme->details		NULL
keyBits	TPMI_RSA_KEY_BITS	2048
exponent	UINT32	0
unique	TPM2B_PUBLIC_KEY_RSA	
size	UINT16	0
buffer	BYTE	Empty

Table 4: Default EK Template (TPMT_PUBLIC) H-1: RSA 2048 (Storage)

B.4.5 Template H-2: ECC NIST P256 (Storage)

Parameter	Type	Content
type	TPMI_ALG_PUBLIC	TPM_ALG_ECC
nameAlg	TPMI_ALG_HASH	TPM_ALG_SHA256
objectAttributes	TPMA_OBJECT	fixedTPM = 1 stClear = 0 fixedParent = 1 sensitiveDataOrigin = 1 userWithAuth = 1 adminWithPolicy = 1 noDA = 0 encryptedDuplication = 0 restricted = 1 decrypt = 1 sign = 0
authPolicy	TPM2B_DIGEST	
size	UINT16	32
buffer	BYTE	0xCA, 0x3D, 0x0A, 0x99, 0xA2, 0xB9, 0x39, 0x06, 0xF7, 0xA3, 0x34, 0x24, 0x14, 0xEF, 0xCF, 0xB3, 0xA3, 0x85, 0xD4, 0x4C, 0xD1, 0xFD, 0x45, 0x90, 0x89, 0xD1, 0x9B, 0x50, 0x71, 0xC0, 0xB7, 0xA0 (Policy B_{SHA256}, see B.6.5)
parameters	TPMS_ECC_PARMS	
symmetric->algorithm	TPMI_ALG_SYM_OBJECT	TPM_ALG_AES
symmetric->keyBits	TPMI_AES_KEY_BITS	128
symmetric->mode	TPMI_SYM_MODE	TPM_ALG_CFB
symmetric->details		NULL
scheme->scheme	TPMI_ALG_ECC_SCHEME	TPM_ALG_NULL
scheme->details		NULL
curveID	TPMI_ECC_CURVE	TPM_ECC_NIST_P256
kdf->scheme	TPMI_ALG_KDF	TPM_ALG_NULL
kdf->details		NULL
unique	TPMS_ECC_POINT	
x->size	UINT16	0
x->buffer	BYTE	Empty
y->size	UINT16	0
y->buffer	BYTE	Empty

Table 5: Default EK Template (TPMT_PUBLIC) H-2: ECC NIST P256 (Storage)

B.4.6 Template H-3: ECC NIST P384 (Storage)

Parameter	Type	Content
type	TPMI_ALG_PUBLIC	TPM_ALG_ECC
nameAlg	TPMI_ALG_HASH	TPM_ALG_SHA384
objectAttributes	TPMA_OBJECT	fixedTPM = 1 stClear = 0 fixedParent = 1 sensitiveDataOrigin = 1 userWithAuth = 1 adminWithPolicy = 1 noDA = 0 encryptedDuplication = 0 restricted = 1 decrypt = 1 sign = 0
authPolicy	TPM2B_DIGEST	
size	UINT16	48
buffer	BYTE	0xB2, 0x6E, 0x7D, 0x28, 0xD1, 0x1A, 0x50, 0xBC, 0x53, 0xD8, 0x82, 0xBC, 0xF5, 0xFD, 0x3A, 0x1A, 0x07, 0x41, 0x48, 0xBB, 0x35, 0xD3, 0xB4, 0xE4, 0xCB, 0x1C, 0x0A, 0xD9, 0xBD, 0xE4, 0x19, 0xCA, 0xCB, 0x47, 0xBA, 0x09, 0x69, 0x96, 0x46, 0x15, 0x0F, 0x9F, 0xC0, 0x00, 0xF3, 0xF8, 0x0E, 0x12 (PolicyB$_{SHA384}$, see B.6.5)
parameters	TPMS_ECC_PARMS	
symmetric->algorithm	TPMI_ALG_SYM_OBJECT	TPM_ALG_AES
symmetric->keyBits	TPMI_AES_KEY_BITS	256
symmetric->mode	TPMI_SYM_MODE	TPM_ALG_CFB
symmetric->details		NULL
scheme->scheme	TPMI_ALG_ECC_SCHEME	TPM_ALG_NULL
scheme->details		NULL
curveID	TPMI_ECC_CURVE	TPM_ECC_NIST_P384
kdf->scheme	TPMI_ALG_KDF	TPM_ALG_NULL
kdf->details		NULL
unique	TPMS_ECC_POINT	
x->size	UINT16	0
x->buffer	BYTE	Empty
y->size	UINT16	0
y->buffer	BYTE	Empty

Table 6: Default EK Template (TPMT_PUBLIC) H-3: ECC NIST P384 (Storage)

NOTE AES 256 bit is used instead of AES 192 bit because a platform specific profile might define AES 192 as optional algorithm.

B.4.7 Template H-4: ECC NIST P521 (Storage)

Parameter	Type	Content
type	TPMI_ALG_PUBLIC	TPM_ALG_ECC
nameAlg	TPMI_ALG_HASH	TPM_ALG_SHA512
objectAttributes	TPMA_OBJECT	fixedTPM = 1 stClear = 0 fixedParent = 1 sensitiveDataOrigin = 1 userWithAuth = 1 adminWithPolicy = 1 noDA = 0 encryptedDuplication = 0 restricted = 1 decrypt = 1 sign = 0
authPolicy	TPM2B_DIGEST	
size	UINT16	64
buffer	BYTE	0xB8, 0x22, 0x1C, 0xA6, 0x9E, 0x85, 0x50, 0xA4, 0x91, 0x4D, 0xE3, 0xFA, 0xA6, 0xA1, 0x8C, 0x07, 0x2C, 0xC0, 0x12, 0x08, 0x07, 0x3A, 0x92, 0x8D, 0x5D, 0x66, 0xD5, 0x9E, 0xF7, 0x9E, 0x49, 0xA4, 0x29, 0xC4, 0x1A, 0x6B, 0x26, 0x95, 0x71, 0xD5, 0x7E, 0xDB, 0x25, 0xFB, 0xDB, 0x18, 0x38, 0x42, 0x56, 0x08, 0xB4, 0x13, 0xCD, 0x61, 0x6A, 0x5F, 0x6D, 0xB5, 0xB6, 0x07, 0x1A, 0xF9, 0x9B, 0xEA ($PolicyB_{SHA512}$, see B.6.5)
parameters	TPMS_ECC_PARMS	
symmetric->algorithm	TPMI_ALG_SYM_OBJECT	TPM_ALG_AES
symmetric->keyBits	TPMI_AES_KEY_BITS	256
symmetric->mode	TPMI_SYM_MODE	TPM_ALG_CFB
symmetric->details		NULL
scheme->scheme	TPMI_ALG_ECC_SCHEME	TPM_ALG_NULL
scheme->details		NULL
curveID	TPMI_ECC_CURVE	TPM_ECC_NIST_P521
kdf->scheme	TPMI_ALG_KDF	TPM_ALG_NULL
kdf->details		NULL
unique	TPMS_ECC_POINT	
x->size	UINT16	0
x->buffer	BYTE	Empty
y->size	UINT16	0
y->buffer	BYTE	Empty

Table 7: Default EK Template (TPMT_PUBLIC) H-4: ECC NIST P521 (Storage)

B.4.8 Template H-5: ECC SM2 P256 (Storage)

Parameter	Type	Content
type	TPMI_ALG_PUBLIC	TPM_ALG_ECC
nameAlg	TPMI_ALG_HASH	TPM_ALG_SM3_256
objectAttributes	TPMA_OBJECT	fixedTPM = 1 stClear = 0 fixedParent = 1 sensitiveDataOrigin = 1 userWithAuth = 1 adminWithPolicy = 1 noDA = 0 encryptedDuplication = 0 restricted = 1 decrypt = 1 sign = 0
authPolicy	TPM2B_DIGEST	
size	UINT16	32
buffer	BYTE	0x16, 0x78, 0x60, 0xA3, 0x5F, 0x2C, 0x5C, 0x35, 0x67, 0xF9, 0xC9, 0x27, 0xAC, 0x56, 0xC0, 0x32, 0xF3, 0xB3, 0xA6, 0x46, 0x2F, 0x8D, 0x03, 0x79, 0x98, 0xE7, 0xA1, 0x0F, 0x77, 0xFA, 0x45, 0x4A ($PolicyB_{SM3\_256}$, see B.6.5)
parameters	TPMS_ECC_PARMS	
symmetric->algorithm	TPMI_ALG_SYM_OBJECT	TPM_ALG_SM4
symmetric->keyBits	TPMI_SM4_KEY_BITS	128
symmetric->mode	TPMI_SYM_MODE	TPM_ALG_CFB
symmetric->details		NULL
scheme->scheme	TPMI_ALG_ECC_SCHEME	TPM_ALG_NULL
scheme->details		NULL
curveID	TPMI_ECC_CURVE	TPM_ECC_SM2_P256
kdf->scheme	TPMI_ALG_KDF	TPM_ALG_NULL
kdf->details		NULL
unique	TPMS_ECC_POINT	
x->size	UINT16	0
x->buffer	BYTE	Empty
y->size	UINT16	0
y->buffer	BYTE	Empty

Table 8: Default EK Template (TPMT_PUBLIC) H-5: SM2 P256 (Storage)

B.4.9 Template H-6: RSA 3072 (Storage)

Parameter	Type	Content
type	TPMI_ALG_PUBLIC	TPM_ALG_RSA
nameAlg	TPMI_ALG_HASH	TPM_ALG_SHA384
objectAttributes	TPMA_OBJECT	fixedTPM = 1 stClear = 0 fixedParent = 1 sensitiveDataOrigin = 1 userWithAuth = 1 adminWithPolicy = 1 noDA = 0 encryptedDuplication = 0 restricted = 1 decrypt = 1 sign = 0
authPolicy	TPM2B_DIGEST	
size	UINT16	48
buffer	BYTE	0xB2, 0x6E, 0x7D, 0x28, 0xD1, 0x1A, 0x50, 0xBC, 0x53, 0xD8, 0x82, 0xBC, 0xF5, 0xFD, 0x3A, 0x1A, 0x07, 0x41, 0x48, 0xBB, 0x35, 0xD3, 0xB4, 0xE4, 0xCB, 0x1C, 0x0A, 0xD9, 0xBD, 0xE4, 0x19, 0xCA, 0xCB, 0x47, 0xBA, 0x09, 0x69, 0x96, 0x46, 0x15, 0x0F, 0x9F, 0xC0, 0x00, 0xF3, 0xF8, 0x0E, 0x12 (PolicyB$_{SHA384}$, see B.6.5)
parameters	TPMS_RSA_PARMS	
symmetric->algorithm	TPMI_ALG_SYM_OBJECT	TPM_ALG_AES
symmetric->keyBits	TPMI_AES_KEY_BITS	256
symmetric->mode	TPMI_SYM_MODE	TPM_ALG_CFB
symmetric->details		NULL
scheme->scheme	TPMI_ALG_ASYM_SCHEME	TPM_ALG_NULL
scheme->details		NULL
keyBits	TPMI_RSA_KEY_BITS	3072
exponent	UINT32	0
unique	TPM2B_PUBLIC_KEY_RSA	
size	UINT16	0
buffer	BYTE	Empty

Table 9: Default EK Template (TPMT_PUBLIC) H-6: RSA 3072 (Storage)

NOTE The selection of SHA384 and AES256 to be used with RSA 3k follows CNSA-Suite recommendations [23].

B.4.10 Template H-7: RSA 4096 (Storage)

Parameter	Type	Content
type	TPMI_ALG_PUBLIC	TPM_ALG_RSA
nameAlg	TPMI_ALG_HASH	TPM_ALG_SHA384
objectAttributes	TPMA_OBJECT	fixedTPM = 1 stClear = 0 fixedParent = 1 sensitiveDataOrigin = 1 userWithAuth = 1 adminWithPolicy = 1 noDA = 0 encryptedDuplication = 0 restricted = 1 decrypt = 1 sign = 0
authPolicy	TPM2B_DIGEST	
size	UINT16	48
buffer	BYTE	0xB2, 0x6E, 0x7D, 0x28, 0xD1, 0x1A, 0x50, 0xBC, 0x53, 0xD8, 0x82, 0xBC, 0xF5, 0xFD, 0x3A, 0x1A, 0x07, 0x41, 0x48, 0xBB, 0x35, 0xD3, 0xB4, 0xE4, 0xCB, 0x1C, 0x0A, 0xD9, 0xBD, 0xE4, 0x19, 0xCA, 0xCB, 0x47, 0xBA, 0x09, 0x69, 0x96, 0x46, 0x15, 0x0F, 0x9F, 0xC0, 0x00, 0xF3, 0xF8, 0x0E, 0x12 (PolicyB$_{SHA384}$, see B.6.5)
parameters	TPMS_RSA_PARMS	
symmetric->algorithm	TPMI_ALG_SYM_OBJECT	TPM_ALG_AES
symmetric->keyBits	TPMI_AES_KEY_BITS	256
symmetric->mode	TPMI_SYM_MODE	TPM_ALG_CFB
symmetric->details		NULL
scheme->scheme	TPMI_ALG_ASYM_SCHEME	TPM_ALG_NULL
scheme->details		NULL
keyBits	TPMI_RSA_KEY_BITS	4096
exponent	UINT32	0
unique	TPM2B_PUBLIC_KEY_RSA	
size	UINT16	0
buffer	BYTE	Empty

Table 10: Default EK Template (TPMT_PUBLIC) H-7: RSA 4096 (Storage)

NOTE The selection of SHA384 and AES256 to be used with RSA 4k follows CNSA-Suite recommendations [23].

B.5 Policy NV Indices

B.5.1 Introduction

This annex defines the public structure (TPMS_NV_PUBLIC) of NV Indices that may be used to store a policy digest in their data (payload) area. This policy may then be used as authorization policy by an Endorsement Key that was created with one of the default EK Templates of the High Range. Other authorizations options – which do not require such an NV Index - are described in B.4.

One NV Index is reserved for each hash algorithm: SHA256, SHA384, SHA512 and SM3_256. Figure 1 illustrates which Policy Index (I-1 to I-4) is to be used by which default EK Template of the High Range (H-1 to H-5).

The Policy NV Indices SHOULD NOT be *Populated* by the TPM manufacturer. Instead, a Policy NV Index MAY be defined and undefined by the platform Owner using Owner Authorization. The NV Index is written with Endorsement authorization.

NOTE It is not anticipated that all four Policy NV Indices would be *Populated* at the same time in a TPM. Instead only the Policy Index corresponding to the EK in operation would be *Populated.* For example Policy Index I-1 for use of EKs created with Template H-1 or H-2.

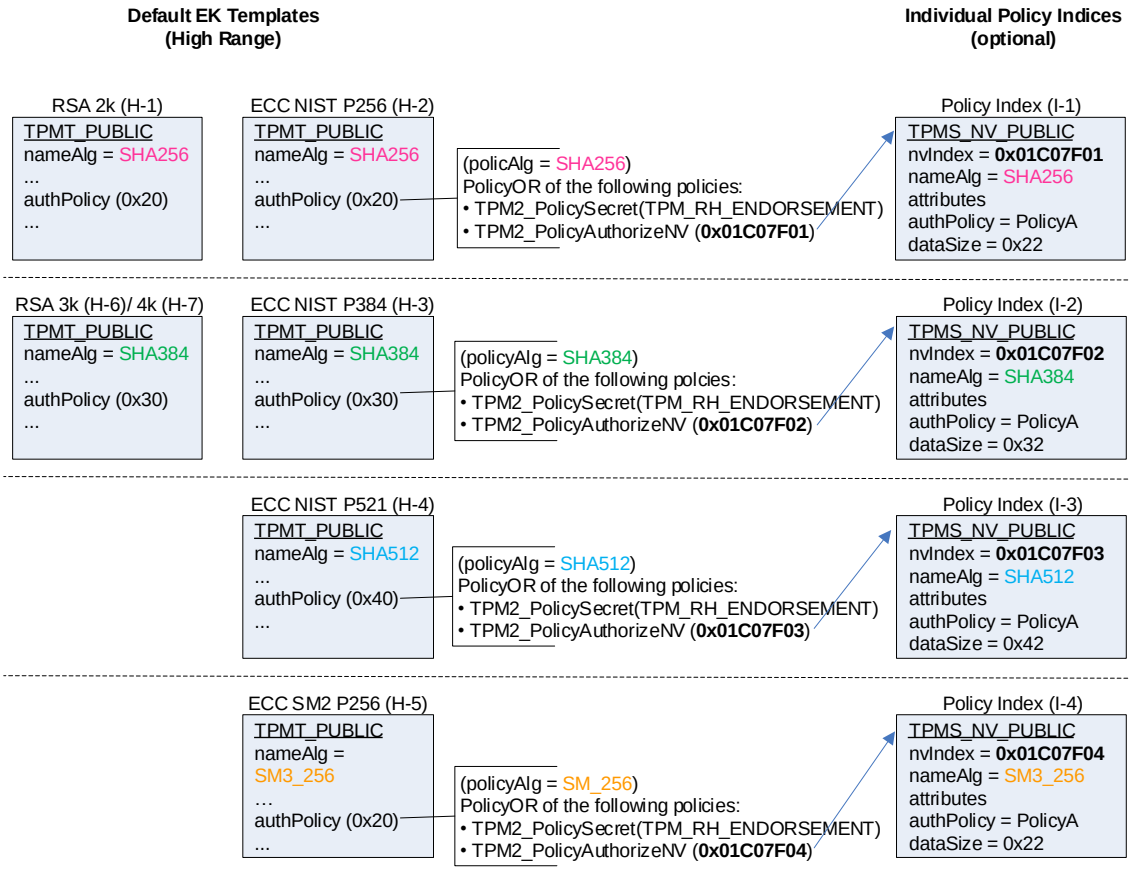

Figure 1: Overview EK Template to Policy Index

B.5.2 Handle Values

The Policy NV Indices (if *Populated*) use the following handle values:

0x01c07f01	Policy Index I-1 with nameAlg = SHA256 (B.5.3)
0x01c07f02	Policy Index I-2 with nameAlg = SHA384 (B.5.4)
0x01c07f03	Policy Index I-3 with nameAlg = SHA512 (B.5.5)
0x01c07f04	Policy Index I-4 with nameAlg = SM3_256 (B.5.6)

B.5.3 Policy Index I-1: SHA256

Parameter	Type	Content
nvIndex	TPMI_RH_NV_INDEX	0x01C07F01
nameAlg	TPMI_ALG_HASH	TPM_ALG_SHA256 (0x000B)
attributes	TPMA_NV	TPMA_NV_PPWRITE = 0 TPMA_NV_OWNERWRITE = 0 TPMA_NV_AUTHWRITE = 0 TPMA_NV_POLICYWRITE = 1 TPM_NT = 0 TPMA_NV_POLICY_DELETE = 0 TPMA_NV_WRITELOCKED = 0 TPMA_NV_WRITEALL = 1 TPMA_NV_WRITEDEFINE = 0 TPMA_NV_WRITE_STCLEAR = 0 TPMA_NV_GLOBALLOCK = 0 TPMA_NV_PPREAD = 1 TPMA_NV_OWNERREAD = 1 TPMA_NV_AUTHREAD = 1 TPMA_NV_POLICYREAD = 1 TPMA_NV_NO_DA = 1 TPMA_NV_ORDERLY = 0 TPMA_NV_CLEAR_STCLEAR = 0 TPMA_NV_READLOCKED = 0 TPMA_NV_WRITTEN = 1 TPMA_NV_PLATFORMCREATE = 0 TPMA_NV_READ_STCLEAR = 0 (0x220F1008)
authPolicy	TPM2B_DIGEST	
size	UINT16	32 (0x0020)
buffer	BYTE	0x83, 0x71, 0x97, 0x67, 0x44, 0x84, 0xB3, 0xF8, 0x1A, 0x90, 0xCC, 0x8D, 0x46, 0xA5, 0xD7, 0x24, 0xFD, 0x52, 0xD7, 0x6E, 0x06, 0x52, 0x0B, 0x64, 0xF2, 0xA1, 0xDA, 0x1B, 0x33, 0x14, 0x69, 0xAA ($PolicyA_{SHA256}$, see Table 15)
dataSize	UINT16	34 (0x0022)

Table 11: EK Policy Index (TPMS_NV_PUBLIC) I-1: SHA256

NOTE The first two bytes of the Index data area contain a TPM_ALG_ID, followed by the policy value (without size).

B.5.4 Policy Index I-2: SHA384

Parameter	Type	Content
nvIndex	TPMI_RH_NV_INDEX	0x01C07F02
nameAlg	TPMI_ALG_HASH	TPM_ALG_SHA384 (0x000C)
attributes	TPMA_NV	TPMA_NV_PPWRITE = 0 TPMA_NV_OWNERWRITE = 0 TPMA_NV_AUTHWRITE = 0 TPMA_NV_POLICYWRITE = 1 TPM_NT = 0 TPMA_NV_POLICY_DELETE = 0 TPMA_NV_WRITELOCKED = 0 TPMA_NV_WRITEALL = 1 TPMA_NV_WRITEDEFINE = 0 TPMA_NV_WRITE_STCLEAR = 0 TPMA_NV_GLOBALLOCK = 0 TPMA_NV_PPREAD = 1 TPMA_NV_OWNERREAD = 1 TPMA_NV_AUTHREAD = 1 TPMA_NV_POLICYREAD = 1 TPMA_NV_NO_DA = 1 TPMA_NV_ORDERLY = 0 TPMA_NV_CLEAR_STCLEAR = 0 TPMA_NV_READLOCKED = 0 TPMA_NV_WRITTEN = 1 TPMA_NV_PLATFORMCREATE = 0 TPMA_NV_READ_STCLEAR = 0 (0x220F1008)
authPolicy	TPM2B_DIGEST	
size	UINT16	48 (0x0030)
buffer	BYTE	0x8B, 0xBF, 0x22, 0x66, 0x53, 0x7C, 0x17, 0x1C, 0xB5, 0x6E, 0x40, 0x3C, 0x4D, 0xC1, 0xD4, 0xB6, 0x4F, 0x43, 0x26, 0x11, 0xDC, 0x38, 0x6E, 0x6F, 0x53, 0x20, 0x50, 0xC3, 0x27, 0x8C, 0x93, 0x0E, 0x14, 0x3E, 0x8B, 0xB1, 0x13, 0x38, 0x24, 0xCC, 0xB4, 0x31, 0x05, 0x38, 0x71, 0xC6, 0xDB, 0x53 (PolicyA$_{SHA384}$, see Table 15)
dataSize	UINT16	50 (0x0032)

Table 12: EK Policy Index (TPMS_NV_PUBLIC) I-2: SHA384

NOTE The first two bytes of the Index data area contain a TPM_ALG_ID, followed by the policy value (without size).

B.5.5 Policy Index I-3: SHA512

Parameter	Type	Content
nvIndex	TPMI_RH_NV_INDEX	0x01C07F03
nameAlg	TPMI_ALG_HASH	TPM_ALG_SHA512 (0x000D)
attributes	TPMA_NV	TPMA_NV_PPWRITE = 0 TPMA_NV_OWNERWRITE = 0 TPMA_NV_AUTHWRITE = 0 TPMA_NV_POLICYWRITE = 1 TPM_NT = 0 TPMA_NV_POLICY_DELETE = 0 TPMA_NV_WRITELOCKED = 0 TPMA_NV_WRITEALL = 1 TPMA_NV_WRITEDEFINE = 0 TPMA_NV_WRITE_STCLEAR = 0 TPMA_NV_GLOBALLOCK = 0 TPMA_NV_PPREAD = 1 TPMA_NV_OWNERREAD = 1 TPMA_NV_AUTHREAD = 1 TPMA_NV_POLICYREAD = 1 TPMA_NV_NO_DA = 1 TPMA_NV_ORDERLY = 0 TPMA_NV_CLEAR_STCLEAR = 0 TPMA_NV_READLOCKED = 0 TPMA_NV_WRITTEN = 1 TPMA_NV_PLATFORMCREATE = 0 TPMA_NV_READ_STCLEAR = 0 (0x220F1008)
authPolicy	TPM2B_DIGEST	
size	UINT16	64 (0x0040)
buffer	BYTE	0x1E, 0x3B, 0x76, 0x50, 0x2C, 0x8A, 0x14, 0x25, 0xAA, 0x0B, 0x7B, 0x3F, 0xC6, 0x46, 0xA1, 0xB0, 0xFA, 0xE0, 0x63, 0xB0, 0x3B, 0x53, 0x68, 0xF9, 0xC4, 0xCD, 0xDE, 0xCA, 0xFF, 0x08, 0x91, 0xDD, 0x68, 0x2B, 0xAC, 0x1A, 0x85, 0xD4, 0xD8, 0x32, 0xB7, 0x81, 0xEA, 0x45, 0x19, 0x15, 0xDE, 0x5F, 0xC5, 0xBF, 0x0D, 0xC4, 0xA1, 0x91, 0x7C, 0xD4, 0x2F, 0xA0, 0x41, 0xE3, 0xF9, 0x98, 0xE0, 0xEE ($PolicyA_{SHA512}$, see Table 15)
dataSize	UINT16	66 (0x0042)

Table 13: EK Policy Index (TPMS_NV_PUBLIC) I-3: SHA512

NOTE The first two bytes of the Index data area contain a TPM_ALG_ID, followed by the policy value (without size).

B.5.6 Policy Index I-4: SM3_256

Parameter	Type	Content
nvIndex	TPMI_RH_NV_INDEX	0x01C07F04
nameAlg	TPMI_ALG_HASH	TPM_ALG_SM3_256 (0x0012)
attributes	TPMA_NV	TPMA_NV_PPWRITE = 0 TPMA_NV_OWNERWRITE = 0 TPMA_NV_AUTHWRITE = 0 TPMA_NV_POLICYWRITE = 1 TPM_NT = 0 TPMA_NV_POLICY_DELETE = 0 TPMA_NV_WRITELOCKED = 0 TPMA_NV_WRITEALL = 1 TPMA_NV_WRITEDEFINE = 0 TPMA_NV_WRITE_STCLEAR = 0 TPMA_NV_GLOBALLOCK = 0 TPMA_NV_PPREAD = 1 TPMA_NV_OWNERREAD = 1 TPMA_NV_AUTHREAD = 1 TPMA_NV_POLICYREAD = 1 TPMA_NV_NO_DA = 1 TPMA_NV_ORDERLY = 0 TPMA_NV_CLEAR_STCLEAR = 0 TPMA_NV_READLOCKED = 0 TPMA_NV_WRITTEN = 1 TPMA_NV_PLATFORMCREATE = 0 TPMA_NV_READ_STCLEAR = 0 (0x220F1008)
authPolicy	TPM2B_DIGEST	
size	UINT16	32 (0x0020)
buffer	BYTE	0xC6, 0x7F, 0x7D, 0x35, 0xF6, 0x6F, 0x3B, 0xEC, 0x13, 0xC8, 0x9F, 0xE8, 0x98, 0x92, 0x1C, 0x65, 0x1B, 0x0C, 0xB5, 0xA3, 0x8A, 0x92, 0x69, 0x0A, 0x62, 0xA4, 0x3C, 0x00, 0x12, 0xE4, 0xFB, 0x8B ($PolicyA_{SM2\_256}$, see Table 15)
dataSize	UINT16	34 (0x0022)

Table 14: EK Policy Index (TPMS_NV_PUBLIC) I-4: SM3_256

NOTE The first two bytes of the Index data area contain a TPM_ALG_ID, followed by the policy value (without size).

B.6 Policy Computation

B.6.1 Introduction

This annex documents how the different policy values used in the default EK Templates (in appendices B.3 and B.4) and the Policy NV Indices (in B.5) were computed. The equations in this section are copied from the TPM 2.0 Library Specification Part 3 [1]. If there are any inconsistencies between the equations below and the equations defined in the TPM 2.0 Library Specification, the definitions in the Library Specification take precedence.

B.6.2 Computing PolicyA

TPM2_PolicySecret() uses the PolicyUpdate function:

PolicyUpdate(TPM_CC_PolicySecret, authObject→Name, policyRef)

This is equivalent to:

$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicySecret || authObject→Name)$
$policyDigest_{new+1} := H_{policyAlg}(policyDigest_{new} || policyRef.buffer)$

With:

policyAlg = SHA256, or SHA384, or SHA512, or SM3_256
$policyDigest_{old}$ = 0x0...0 (32, or 48, or 64 bytes)
TPM_CC_PolicySecret = 0x00000151
authObject→Name is TPM_RH_ENDORSEMENT (=0x4000000B)
policyRef.buffer = not available (policyRef is an Empty Buffer)

The policy digest is calculated as follows:

$policyDigest_{new} := H_{policyAlg}(0x0...0 || 0x00000151 || 0x4000000B)$
$policyDigest_{new+1} := H_{policyAlg}(policyDigest_{new})$

The following table contains the computed *PolicyA* values for the different hash algorithms.

PolicyA	Value (hex)
$PolicyA_{SHA256}$ $H_{SHA256}(H_{SHA256}(0x0...0 \|\| 0x00000151 \|\| 0x4000000B))$	837197674484b3f81a90cc8d46a5d724 fd52d76e06520b64f2a1da1b331469aa
$PolicyA_{SHA384}$ $H_{SHA384}(H_{SHA384}(0x0...0 \|\| 0x00000151 \|\| 0x4000000B))$	8bbf2266537c171cb56e403c4dc1d4b6 4f432611dc386e6f532050c3278c930e 143e8bb1133824ccb431053871c6db53
$PolicyA_{SHA512}$ $H_{SHA512}(H_{SHA512}(0x0...0 \|\| 0x00000151 \|\| 0x4000000B))$	1e3b76502c8a1425aa0b7b3fc646a1b0 fae063b03b5368f9c4cddecaff0891dd 682bac1a85d4d832b781ea451915de5f c5bf0dc4a1917cd42fa041e3f998e0ee
$PolicyA_{SM3\_256}$ $H_{SM3\_256}(H_{SM3\_256}(0x0...0 \|\| 0x00000151 \|\| 0x4000000B))$	c67f7d35f66f3bec13c89fe898921c65 1b0cb5a38a92690a62a43c0012e4fb8b

Table 15: PolicyA values

B.6.3 Computing Policy Index Names

The NV Index Name is computed as defined in Part 1:

Name ≔ nameAlg || $H_{nameAlg}$ (handle→nvPublicArea):

Where

handle→nvPublicArea is the public area of the NV Index (TPMS_NV_PUBLIC) and
TPMS_NV_PUBLIC = nvIndex || nameAlg || attributes || authPolicy || dataSize

With:

nvIndex = 0x01C07F01, 0x01C07F02, 0x01C07F03, or 0x01C07F04
nameAlg = TPM_ALG_SHA256/ SHA384/ SHA512, or TPM_ALG_SM2
attributes = 0x220F1008
authPolicy = PolicyA (including size), see Table 15
dataSize = 0x0022, 0x0032, or 0x0042

The name for the Policy Indices I-1 to I-4 (see B.5.3 to B.5.6) is computed as shown in Table 16.

Policy Index Names	Value (hex)
I-1 Name 0x000B || H_{SHA256}(0x01C07F01 || 0x000B || 0x220F1008 || 0x0020 || $PolicyA_{SHA256}$ || 0x0022)	000b0c9d717e9c3fe69fda41769450bb145957f8b3610e084dbf65591a5d11ecd83f
I-2 Name 0x000C || H_{SHA384}(0x01C07F02 || 0x000C || 0x220F1008 || 0x0030 || $PolicyA_{SHA384}$ || 0x0032)	000cdb62fca346612c976732ff4e8621fb4e858be82586486504f7d02e621f8d7d61ae32cfc60c4d120609ed6768afcf090c
I-3 Name 0x000D || H_{SHA512}(0x01C07F03 || 0x000D || 0x220F1008 || 0x0040 || $PolicyA_{SHA512}$ || 0x0042)	000d1c47c0bbcbd3cf7d7cae6987d31937c171015dde3b7f0d3c869bca1f7e8a223b9acfadb49b7c9cf14d450f41e9327de34d9291eece2c58ab1dc10e9059cce560
I-4 Name 0x0012 || H_{SM3_256}(0x01C07F04 || 0x0012 || 0x220F1008 || 0x0020 || $PolicyA_{SM3\_256}$ || 0x0022)	001298c4652e788dd7ddcccc353a5ea1a0e0b5efd2e7af1afb09cae8d9453c5f1152

Table 16: Policy Index Names

B.6.4 Computing PolicyC

The policy digest for TPM2_PolicyAuthorizeNV() is computed as defined in the Library Spec, Part 3:

$policyDigest_{new}$ ≔ $H_{policyAlg}$($policyDigest_{old}$ || TPM_CC_PolicyAuthorizeNV || nvIndex→Name)

Where

policyAlg = SHA256, SHA384, SHA512, or SM3_256
$policyDigest_{old}$ = 0x0...0 (32, 48, or 64 bytes)
TPM_CC_ PolicyAuthorizeNV = 0x00000192
nvIndex→Name is the Name of the NV Index containing the policy

With the Policy Index Names from Table 16, Policy C is computed as shown in Table 17.

PolicyC	Value (in hex)
PolicyC$_{SHA256}$ H_{SHA256}(0x0...0 || 0x00000192 || Name of Policy Index I-1)	3767e2edd43ff45a3a7e1eaefcef7864 3dca964632e7aad82c673a30d8633fde
PolicyC$_{SHA384}$ H_{SHA384}(0x0...0 || 0x00000192 || Name of Policy Index I-2)	d6032ce61f2fb3c240eb3cf6a33237ef 2b6a16f4293c22b455e261cffd217ad5 b4947c2d73e63005eed2dc2b3593d165
PolicyC$_{SHA512}$ H_{SHA512}(0x0...0 || 0x00000192 || Name of Policy Index I-3)	589ee1e146544716e8deafe6db247b01 b81e9f9c7dd16b814aa159138749105f ba5388dd1dea702f35240c184933121e 2c61b8f50d3ef91393a49a38c3f73fc8
PolicyC$_{SM3\_256}$ H_{SM3_256}(0x0...0 || 0x00000192 || Name of Policy Index I-4)	2d4e81578c3531d9bd1cdd7d02ba298d 5699a3e39fc3551bfeffcf132b49e11d

Table 17: PolicyC values

B.6.5 Computing PolicyB

The policy digest for TPM2_PolicyOR() is computed as defined in the Library Spec, Part 3:

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, TPM\_CC\_PolicyOR \,||\, digests)$$

Where

digests := pHashList.digests[1].buffer || … || pHashList.digests[n].buffer

With

TPM_CC_PolicyOR = 0x00000171
pHashList.digests[1].buffer = PolicyA from Table 15
pHashList.digests[2].buffer = PolicyC from Table 17

PolicyB is computed as shown in Table 18.

PolicyB	Value (in hex)
PolicyB$_{SHA256}$ H_{SHA256}(0x0...0 || 0x00000171 || *PolicyA*$_{SHA256}$ || *PolicyC*$_{SHA256}$)	ca3d0a99a2b93906f7a3342414efcfb3 a385d44cd1fd459089d19b5071c0b7a0
PolicyB$_{SHA384}$ H_{SHA384}(0x0...0 || 0x00000171 || *PolicyA*$_{SHA384}$ || *PolicyC*$_{SHA384}$)	b26e7d28d11a50bc53d882bcf5fd3a1a 074148bb35d3b4e4cb1c0ad9bde419ca cb47ba09699646150f9fc000f3f80e12
PolicyB$_{SHA512}$ H_{SHA512}(0x0...0 || 0x00000171 || *PolicyA*$_{SHA512}$ || *PolicyC*$_{SHA512}$)	b8221ca69e8550a4914de3faa6a18c07 2cc01208073a928d5d66d59ef79e49a4 29c41a6b269571d57edb25fbdb183842 5608b413cd616a5f6db5b6071af99bea
PolicyB$_{SM3\_256}$ H_{SM3_256}(0x0...0 || 0x00000171 || *PolicyA*$_{SM3\_256}$ || *PolicyC*$_{SM3\_256}$)	167860a35f2c5c3567f9c927ac56c032 f3b3a6462f8d037998e7a10f77fa454a

Table 18: PolicyB values

C. Certificate Fields (algorithm-specific)

C.1 Signature Algorithm

The signature algorithm depends on the algorithm of the CA key used to sign the EK certificate. The security strength of the signing algorithm SHOULD be equivalent to the security strength of the signing key. The security strength of the CA key used to sign the EK certificate SHALL have an equal or higher security strength than the EK. An EK certificate for an RSA EK MAY be signed using ECDSA with an ECC NIST P256, 384, or 521 CA Key.

C.1.1 RSA

When using an RSA CA key, the EK certificate SHOULD be signed using the algorithms appropriate to the key size. The AlgorithmIdentifier parameters field MUST be the ASN.1 type NULL.

C.1.1.1 RSA 2k CA Key

For an RSA 2k CA key, the algorithm SHOULD be sha256WithRSAEncryption as defined in RFC 5754 [13].

```
sha256WithRSAEncryption OBJECT IDENTIFIER  ::=  {
    iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 11 }
```

C.1.1.2 RSA 3k and 4k CA Key

For an RSA 3k or 4k CA key, the algorithm SHOULD be sha384WithRSAEncryption as defined in RFC 5754 [13].

```
sha384WithRSAEncryption  OBJECT IDENTIFIER  ::=  {
    iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 12 }
```

NOTE 1 The use of SHA384 with RSA 3k and 4k follows CNSA-Suite recommendations [23].

NOTE 2 The recommendation for use of sha384WithRSAEncryption for RSA 3k and 4k CA key was added in version 2.3 of this specification.

C.1.2 ECC

When using an ECC CA key, the EK certificate SHOULD be signed using the algorithms appropriate to the curve size. The AlgorithmIdentifier parameters field MUST be absent.

C.1.2.1 NIST P256 CA Key

For an ECC NIST P256 CA key, the algorithm SHOULD be ecdsa-with-SHA256 as defined in RFC 5754 [13].

```
ecdsa-with-SHA256 OBJECT IDENTIFIER  ::=  { iso(1) member-body(2)
    us(840)ansi-X9-62(10045) signatures(4) ecdsa-with-SHA2(3) 2 }
```

C.1.2.2 NIST P384 CA Key

For an ECC NIST P384 CA key, the algorithm SHOULD be ecdsa-with-SHA384 as defined in RFC 5754 [13].

```
ecdsa-with-SHA384 OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) ansi-X9-62(10045) signatures(4) ecdsa-with-SHA2(3) 3 }
```

C.1.2.3 NIST P521 CA Key

For an ECC NIST P521 CA key, the algorithm SHOULD be ecdsa-with-SHA512 as defined in RFC 5754 [13].

```
ecdsa-with-SHA512 OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) ansi-X9-62(10045) signatures(4) ecdsa-with-SHA2(3) 4 }
```

C.1.2.4 SM2 P256 CA Key

When using an ECC SM2 key, the EK certificate SHOULD be signed using the algorithm SM3WithSM2Encryption which has the OID value defined in GM/T 0006-2012 Cryptographic

Application Identifier Criterion Specification [20] as shown below. The AlgorithmIdentifier parameters field MUST be absent.

```
SM3WithSM2Encryption OBJECT IDENTIFIER  ::=  { iso(1) member-body(2) cn(156)
    ccstc(10197) cryptographic-algorithm (1) 501 }
```

C.2 Subject Public Key Info

C.2.1 RSA

For an RSA public key the algorithm rsaEncryption that has the OID value defined in RFC 3279 [14] as shown below MUST be used. The AlgorithmIdentifier parameters field MUST be the ASN.1 type NULL.

```
rsaEncryption OBJECT IDENTIFIER  ::=  {
    iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 1 }
```

The RSA public key MUST be encoded using the ASN.1 type RSAPublicKey as defined in RFC 3279 [14].

```
RSAPublicKey ::= SEQUENCE {
    modulus            INTEGER,    -- n
    publicExponent     INTEGER  }  -- e
```

C.2.2 ECC

For an ECC public key the algorithm id-ecPublicKey which has the OID value defined in RFC 5480 [15] as shown below MUST be used. The ECParameters field is required, the nameCurve field SHOULD contain the OID of the respective curve (see below).

```
id-ecPublicKey OBJECT IDENTIFIER  ::=  {
    iso(1) member-body(2) us(840) ansi-X9-62(10045) keyType(2) 1 }


ECParameters ::= CHOICE {
    namedCurve         OBJECT IDENTIFIER
    -- implicitCurve   NULL
    -- specifiedCurve  SpecifiedECDomain
    }
```

The ECC public key MUST be encoded as an ECC Point. The uncompressed format SHOULD be used.

```
ECPoint ::= OCTET STRING
```

The namedCurve field in ECParameters of the Subject Public Key Info depends on the ECC curve.

C.2.2.1 NIST P256

For NIST P256, the namedCurve field MUST contain the OID defined in RFC 5480 [15]:

```
secp256r1 OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) ansi-X9-62(10045) curves(3) prime(1) 7 }
```

C.2.2.2 NIST P384

For NIST P384, the namedCurve field MUST contain the OID defined in RFC 5480 [15]:

```
secp384r1 OBJECT IDENTIFIER ::= {
    iso(1) identified-organization(3) certicom(132) curve(0) 34 }
```

C.2.2.3 NIST P521

For NIST P521, the namedCurve field MUST contain the OID defined in RFC 5480 [15]:

```
secp521r1 OBJECT IDENTIFIER ::= {
    iso(1) identified-organization(3) certicom(132) curve(0) 35 }
```

C.2.2.4 SM2 P256

For SM2 P256, the namedCurve field MUST contain the OID defined in GM/T 0006-2012 Cryptographic Application Identifier Criterion Specification [20]:

```
SM2EllipticCurveCryptography OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) cn(156) ccstc(10197) cryptographic-algorithm (1) 301 }
```