Proposal:
Interoperable Attested TLS

Presenter: Shanwei Cen

For CCC Attestation SIG

August 30 / September 13, 2022

Outline

- Existing Projects
- Objectives
- The Proposal
- Supported Usages

Existing RA-TLS Libraries

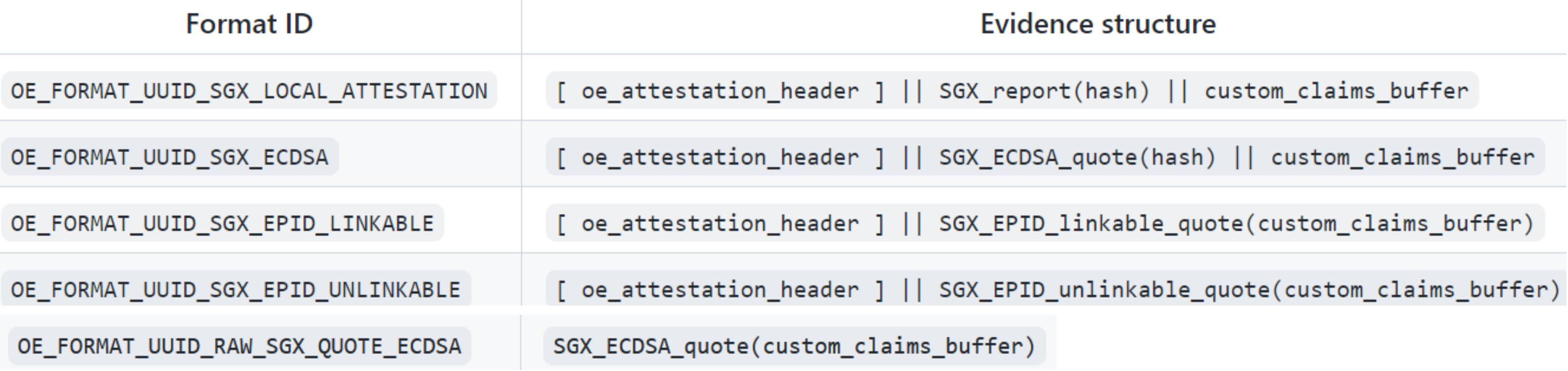
- Issues
  - No interoperability between different RA-TLS Libraries
    - Each library assigns their own X.509 cert extension OIDs for their own (slightly different) evidence formats
    - Each project implements its own cert generation and verification libraries.
  - RA-TLS evidence freshness not bound to the TLS session

	RA-TLS (Gramine)	OE Attested TLS (OE: Open Enclave SDK)	RATS-TLS
X.509 Extension	One Intel OID, for SGX ECDSA quote (with SHA256 hash of the public key held in SGX report user data)	Two Microsoft OIDs • One for raw SGX ECDSA quote (pubkey itself, not hash, is held in SGX report user data field) • The other for OE evidence: header + quote + pubkey (its value held as custom_claims_buffer)	Multiple Intel OIDs, one for each TEE / evidence format. e.g. • ecdsa_quote_oid (same OID and quote format as RA-TLS • la_report_oid: SGX report for local attestation • tdx_quote_oid • sev_report_oid
Cert / evidence generation	* Get quote from Gramine psuedo file /dev/attestation/quote * Create cert with library API ra_tls_create_key_and_cert_der() etc.	* Get evidence from OE SDK oe_get_evidence() * Create cert with function oe_gen_custom_x509_cert()	* Get ECDSA quote from DCAP library. * Create cert in the attester wrapper
Cert / evidence verification	* TLS library callback function ra_tls_verify_callback() implemented to verify cert, get OID and verify evidence, and verify that cert public key is covered by evidence	* Similar to RA-TLS * calls oe_verify_evidence(). * Different callback functions for different TLS libs	* Set callback function to TLS libraries for cert and evidence verification

OE SDK Attested-TLS Evidence

Evidence formats

- Little-endian data
- EPID formats does not support per-session freshness, due to limit in custom claims buffer size.
- Source: Attestation API Proposal.md



Format ID	Evidence structure
`OE_FORMAT_UUID_SGX_LOCAL_ATTESTATION`	`[ oe_attestation_header ] || SGX_report(hash) || custom_claims_buffer`
`OE_FORMAT_UUID_SGX_ECDSA`	`[ oe_attestation_header ] || SGX_ECDSA_quote(hash) || custom_claims_buffer`
`OE_FORMAT_UUID_SGX_EPID_LINKABLE`	`[ oe_attestation_header ] || SGX_EPID_linkable_quote(custom_claims_buffer)`
`OE_FORMAT_UUID_SGX_EPID_UNLINKABLE`	`[ oe_attestation_header ] || SGX_EPID_unlinkable_quote(custom_claims_buffer)`
`OE_FORMAT_UUID_RAW_SGX_QUOTE_ECDSA`	`SGX_ECDSA_quote(custom_claims_buffer)`

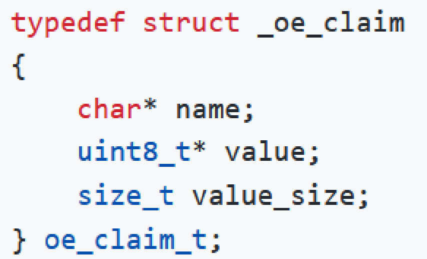
API and Header structure

- API: attester.h verifier.h
- Header: attest_plug.h

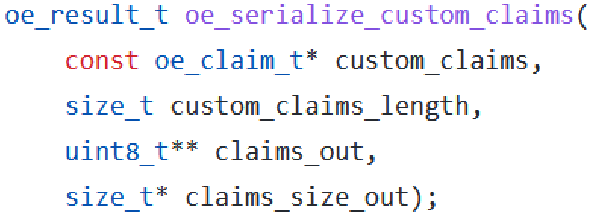
- API oe_get_evidence() Input parameter flags=0 results in output evidence and (optional) endorsements without OE attestation header.
- API oe_verify_evidence() can also take evidence input as two parameters: UUID and evidence (without header). It also takes endorsements

Custom claims serialization

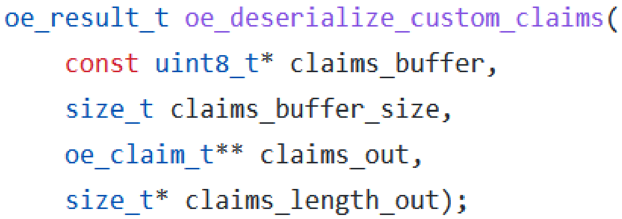
- Name: null-terminated string
- Value: binary byte array
- Source custom_claims.h



```
typedef struct _oe_claim
{
    char* name;
    uint8_t* value;
    size_t value_size;
} oe_claim_t;
```



```
oe_result_t oe_serialize_custom_claims(
    const oe_claim_t* custom_claims,
    size_t custom_claims_length,
    uint8_t** claims_out,
    size_t* claims_size_out);
```



```
oe_result_t oe_deserialize_custom_claims(
    const uint8_t* claims_buffer,
    size_t claims_buffer_size,
    oe_claim_t** claims_out,
    size_t* claims_length_out);
```

Endorsement serialization

- Definition: RemoteAttestationCollaterals.md

Attested TLS specifics

- Self-signed cert, keypair algorithm EC_SECP256P1, in PEM format
- custom_claims_buffer holds the public key value as a byte-array, no algorithm ID info

Objectives

- Interoperability between different RA-TLS libraries
    - So different libraries can be used on either end of a TLS session
- Support of per-session evidence freshness
    - So verifier can be sure the evidence is generated for the current session
- Work with existing TLS protocol versions
- Align with relevant existing standards
- Readily extensible to support new TEEs and evidence formats

The Proposal: Definition

- Adopt standard TCG DICE X.509 cert extension OIDs and evidence / endorsement formats
  - Evidence: X.509 cert extension, as cbor-tagged bstr
    - OID: tcg-dice-tagged-evidence (2.23.133.4.9)
    - IANA-registered cbor tag as evidence format ID
      - Registered with definition of evidence format and procedures for its generation and verification
    - Evidence data (including custom claims) as bstr
  - Endorsement (optional): X.509 cert extension, as cbor-tagged bstr
    - OID for cbor-tagged endorsement bstr: tcg-dice-endorsement-manifest (2.23.133.4.2)
      - Manifest format: cbor-tagged bstr
      - cbor tag: same as that for evidence extension
- Add TEE evidence support of per-session freshness
  - Evidence inclusion of peer nonce in the TLS handshake, in addition to cert public key

Cert and Evidence Formats

- X.509 cert with the proposed evidence and (optional) endorsements extensions
  - Cert can be self-signed or signed by a CA
    - Note: CA-signed cert asserts both the attester TCB and ownership
  - Works with existing TLS protocol versions
- Evidence claims
  - “pubkey” (required): holds cert subject public key value as byte array
    - Note: this claim does not include information like algorithm ID, which is already in the cert.
  - “nonce” (optional): holds nonce as a byte array
    - Note: if a use case does not need pre-session freshness, this claim is not present.
- SGX evidence Formats
  - SGX ECDSA evidence, two options
    - Self-contained quote: **SGX_ECDSA_quote(pubkey-value)**
      - The public key value itself, not its hash, is held in the enclave SGX report user data field
    - Quote with serialized claims : **SGX_ECDSA_quote(hash) + claims-buffer**
      - Claims-buffer holds serialized evidence claims, can support both pubkey and nonce
  - SGX Local attestation, two options
    - Self-contained SGX report: **SGX_report(pubkey-value)**
    - SGX report with serialized claims: **SGX_report(hash) + claims-buffer**
  - SGX EPID evidence: **SGX_EPID_quote(pubkey-value)**
- New Evidence formats: TDX, SEV-SNP, etc.

SGX Evidence and Endorsement Formats

- Claims-buffer holds a serialized cbor map of one or two claims
  - “pubkey” (required)
  - “nonce” (optional)
- Evidence extension OID data as a cbor-tagged array of one or two entries
  - First entry (required): SGX quote or report
  - Second entry (optional): claims-buffer
    - If not present, SGX report user data field contains public key value.
- Endorsement extension OID data as a cbor-tagged array of 9 entries (each a bstr)
  - Only supported for ECDSA quote
  - Index of each entry defined in OE SDK oe_sgx_endorsements_fields_t in bits/attestation.h: VERSION, TCB_INFO, TCB_ISSUER_CHAIN, CRL_PCK_CERT, CRL_PCK_PROC_CA, CRL_ISSUER_CHAIN_PCK_CERT, QE_ID_INFO, QE_ID_ISSUER_CHAIN, CREATION_DATETIME

Example cbor data:

```
{ "pubkey" : h'44e0cad0fc96f42869b816622eb110bd’,
  "nonce" : h'34283f541b4dc2dd0d4849ee644ef166'
}
```

```
4711([
h'6eb527fb1dc82eff5665ab3c63403937d00ab0f9ccd3417cb22b75d40dfc5ae5a
306b8d8ed7e239c2f97d5e242a8f83c721e4d6209919c2d918fa07ad28445c0',
h'b1c16c12845f164be2d4fec22c67f852cacd1d4b9bd8020c6ecd00254703c900'
])
```

```
4711([
h'0000', h'1111', h'2222', h'3333', h'4444', h'5555', h'6666', h'7777', h'8888'
])
```

Example CBOR Data and Serialization

Example claims-buffer:

```
{ "pubkey" : h'44e0cad0fc96f42869b816622eb110bd',
  "nonce" : h'34283f541b4dc2dd0d4849ee644ef166'
}
```

➔

```
A2                                     # map(2)
   66                                  # text(6)
      7075626B6579                     # "pubkey"
   50                                  # bytes(16)
      44E0CAD0FC96F42869B816622EB110BD # "D\xE0\xCA\xD0\xFC\x96\xF4(i
\xB8\u0016b.\xB1\u0010\xBD"
   65                                  # text(5)
      6E6F6E6365                       # "nonce"
   50                                  # bytes(16)
      34283F541B4DC2DD0D4849EE644EF166 # "4(?T\eM\xC2\xDD\rHI\xEEdN\xF1f"
```

Example evidence extension OID data:

```
4711([
h'6eb527fb1dc82eff5665ab3c63403937d00ab0f9ccd3417cb22b75d40dfc5ae5a
306b8d8ed7e239c2f97d5e242a8f83c721e4d6209919c2d918fa07ad28445c0',
h'b1c16c12845f164be2d4fec22c67f852cacd1d4b9bd8020c6ecd00254703c900'
])
```

➔

```
D9 1267                                  # tag(4711)
   82                                    # array(2)
      58 40                              # bytes(64)

6EB527FB1DC82EFF5665AB3C63403937D00AB0F9CCD3417CB22B75D40DFC5AE5A306B8D8E
D7E239C2F97D5E242A8F83C721E4D6209919C2D918FA07AD28445C0 # "n\xB5'
\xFB\u001D\xC8.\xFFVe\xAB<c@97\xD0\n\xB0\xF9\xCC\xD3A|\xB2+u\xD4\r
\xFCZ\xE5\xA3\u0006\xB8\xD8\xED~#\x9C/\x97\xD5\xE2B\xA8\xF8<r\u001EMb
\t\x91\x9C-\x91\x8F\xA0žE\xC0"
      58 20                              # bytes(32)
         B1C16C12845F164BE2D4FEC22C67F852CACD1D4B9BD8020C6ECD00254703C900
# "\xB1\xC1l\u0012\x84_\u0016K\xE2\xD4\xFE\xC2,g\xF8R\xCA\xCD\u001DK
\x9B\xD8\u0002\fn\xCD\u0000%G\u0003\xC9\u0000"
```

Example endorsement extension OID data:

```
4711([
h'0000', h'1111', h'2222', h'3333', h'4444', h'5555', h'6666', h'7777', h'8888'
])
```

➔

```
D9 1267       # tag(4711)
   89         # array(9)
      42      # bytes(2)                          42      # bytes(2)
         0000 # "\u0000\u0000"                       5555 # "UU"
      42      # bytes(2)                          42      # bytes(2)
         1111 # "\u0011\u0011"                       6666 # "ff"
      42      # bytes(2)                          42      # bytes(2)
         2222 # "\"\""                               7777 # "WW"
      42      # bytes(2)                          42      # bytes(2)
         3333 # "33"                                 8888 # "\x88\x88"
      42      # bytes(2)
         4444 # "DD"
```

Note: serialization with CBOR playground

The Proposal: Implementation

- Add support of callback functions in TLS Libraries (openssl, mbedtls, wolftls, etc.) for evidence / cert generation and verification
  - Add attester callback functions, to input peer nonce, and output generated cert
    - Evidence / cert could be generated in advance if per-session freshness is not required. In this case, nonce claim is not present in the evidence.
  - Modify verifier callback functions, to input both the received cert and its own session nonce, for verification of evidence and its freshness
    - Callback implementation can ignore checking the nonce claim in evidence, if it does not require per-session freshness
- Implement common library for cert and evidence generation and verification
  - Claims, evidence, endorsement serialization and de-serialization functions
  - TLS libraries callback functions for X.509 cert generation and verification
    - Different implementation for different TLS libraries.
    - With hooks to TEE-specific implementation for evidence generation and verification

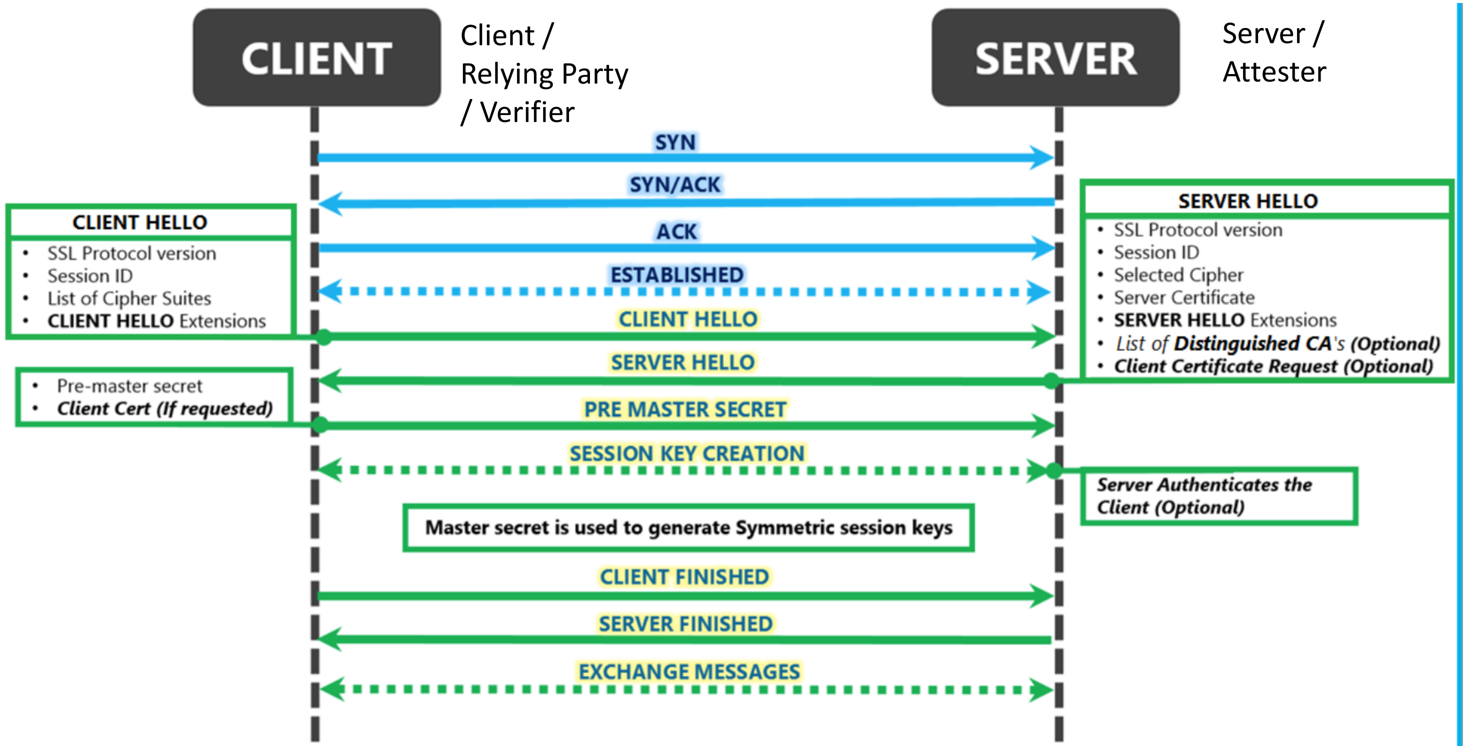
Note: CCC Projects and Submission Form

Supported Usages

- Interoperability between different RA-TLS based libraries conforming to the cert and evidence format definition:
  - Gramine: change implementation to proposed OIDs and format for SGX ECDSA quote, with two custom claims. Reuse TEE-agnostic library
  - OE: change RA-TLS implementation to use proposed OIDs and evidence formats. Evidence to include the two custom claims when possible. Reuse TEE-agnostic library.
  - RATS-TLS: change implementation to proposed OIDs and formats for SGX, TDX, SEV, etc.
- The cert and evidence definition can be used outside of the TLS context
  - MAA JWK signing cert?
  - Evidence wrapper for STET, HTTPA etc.?

Backup

TLS Handshake Primer



CLIENT

Client / Relying Party / Verifier

SERVER

Server / Attester

CLIENT HELLO

- SSL Protocol version
- Session ID
- List of Cipher Suites
- **CLIENT HELLO** Extensions

SERVER HELLO

- SSL Protocol version
- Session ID
- Selected Cipher
- Server Certificate
- **SERVER HELLO** Extensions
- *List of **Distinguished CA**'s **(Optional)***
- ***Client Certificate Request (Optional)***

- Pre-master secret
- ***Client Cert (If requested)***

Server Authenticates the Client (Optional)

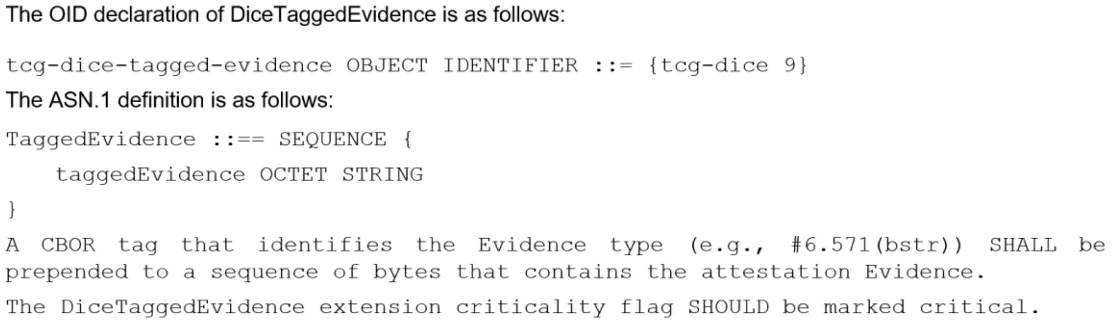
Master secret is used to generate Symmetric session keys

- Pre-master = derive(server.key_exchange_params, client.key_exchange_params)
- Master secret = derive(client.random, server.random, pre-master)

Source: white paper: SSL server authentication and SSL Handshake

TCG DICE Evidence and Endorsement Extensions

Source: TCG DICE Attestation Architecture v1.1 r6 [specification]



The OID declaration of DiceTaggedEvidence is as follows:

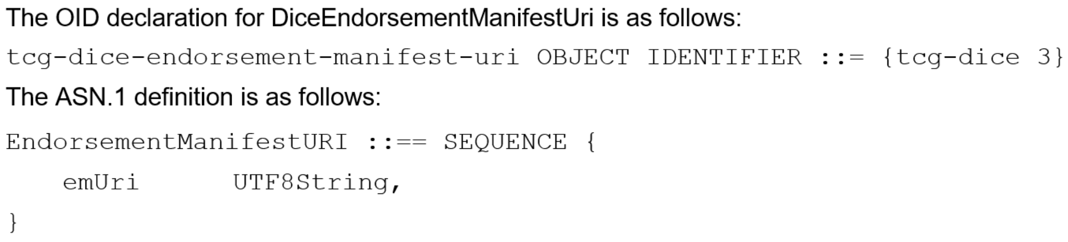
```
tcg-dice-tagged-evidence OBJECT IDENTIFIER ::= {tcg-dice 9}
```

The ASN.1 definition is as follows:

```
TaggedEvidence ::== SEQUENCE {
    taggedEvidence OCTET STRING
}
```

`A CBOR tag that identifies the Evidence type (e.g., #6.571(bstr)) SHALL be prepended to a sequence of bytes that contains the attestation Evidence.`

`The DiceTaggedEvidence extension criticality flag SHOULD be marked critical.`

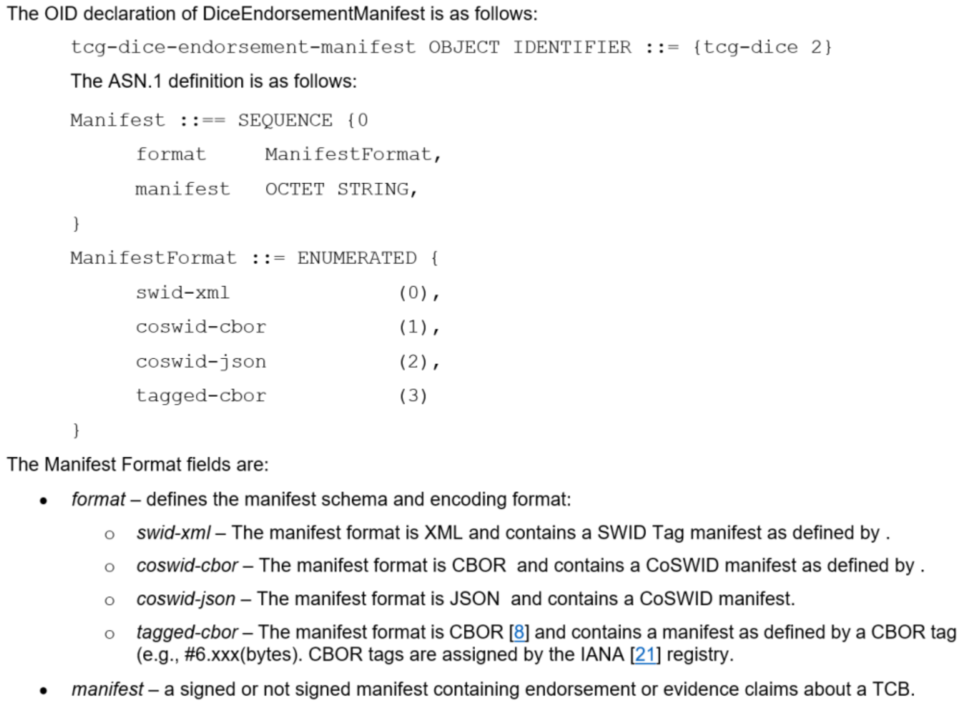


The OID declaration for DiceEndorsementManifestUri is as follows:

```
tcg-dice-endorsement-manifest-uri OBJECT IDENTIFIER ::= {tcg-dice 3}
```

The ASN.1 definition is as follows:

```
EndorsementManifestURI ::== SEQUENCE {
    emUri      UTF8String,
}
```



The OID declaration of DiceEndorsementManifest is as follows:

```
tcg-dice-endorsement-manifest OBJECT IDENTIFIER ::= {tcg-dice 2}
```

The ASN.1 definition is as follows:

```
Manifest ::== SEQUENCE {0
      format      ManifestFormat,
      manifest    OCTET STRING,
}
ManifestFormat ::= ENUMERATED {
      swid-xml              (0),
      coswid-cbor           (1),
      coswid-json           (2),
      tagged-cbor           (3)
}
```

The Manifest Format fields are:

- *format* – defines the manifest schema and encoding format:
  - *swid-xml* – The manifest format is XML and contains a SWID Tag manifest as defined by .
  - *coswid-cbor* – The manifest format is CBOR and contains a CoSWID manifest as defined by .
  - *coswid-json* – The manifest format is JSON and contains a CoSWID manifest.
  - *tagged-cbor* – The manifest format is CBOR [8] and contains a manifest as defined by a CBOR tag (e.g., #6.xxx(bytes). CBOR tags are assigned by the IANA [21] registry.
- *manifest* – a signed or not signed manifest containing endorsement or evidence claims about a TCB.