# Objects

Cryptoki recognizes a number of classes of objects, as defined in the **CK\_OBJECT\_CLASS** data type. An object consists of a set of attributes, each of which has a given value. Each attribute that an object possesses has precisely one value. The following figure illustrates the high-level hierarchy of the Cryptoki objects and some of the attributes they support:



*<In the above table, trust objects are a subclass of Storage>*

Insert after 4.6 Certificate objects:

## Trust objects

### Definitions

This section defines the object class CKO\_TRUST for type CK\_OBJECT\_CLASS as used in the CKA\_CLASS attribute of objects.

CK\_TRUST is defined as:

typedef CK\_ULONG CK\_TRUST

and can have the following values: CKT\_TRUSTED, CKT\_TRUSTED\_DELEGATOR, CKT\_NOT\_TRUSTED, CKT\_TRUST\_MUST\_VERIFY\_TRUST, or CKT\_TRUST\_UNKNOWN.

### Overview

Trust objects (object class **CKO\_TRUST**) bind trusted usages to individual certificates. Trust objects for a given certificate are accessed with the CKA\_ISSUER and CKA\_SERIAL\_NUMBER. The corresponding certificate does not necessarily have to exist in the same token as its trust object. Multiple trust objects for the same certificate can exist in different tokens, but each token should have only one trust object for a given certificate.

Table 1, Trust Object Attributes

| **Attribute** | **Data type** | **Meaning** |
| --- | --- | --- |
| CKA\_ISSUER1 | Byte Array | DER-encoding of the attribute certificate's issuer field. This is distinct from the CKA\_ISSUER attribute contained in CKC\_X\_509 certificates because the ASN.1 syntax and encoding are different. (default empty) |
| CKA\_SERIAL\_NUMBER1 | Byte Array | DER-encoding of the certificate serial number. (default empty) |
| CKA\_HASH\_OF\_CERTIFICATE2 | Byte array | SHA-1 hash of the certificate (default empty). Hash algorithm is defined by CKA\_NAME\_HASH\_ALGORITHM |
| CKA\_NAME\_HASH\_ALGORITHM2 | CK\_MECHANISM\_TYPE | Defines the mechanism used to calculate CKA\_HASH\_OF\_SUBJECT\_PUBLIC\_KEY and CKA\_HASH\_OF\_ISSUER\_PUBLIC\_KEY. If the attribute is not present then the type defaults to SHA-1. |
| CKA\_TRUST\_SERVER\_AUTH3 | CK\_TRUST | Trusted for authenticating a server in a client/server module (example TLS/SSL/SSH) |
| CKA\_TRUST\_CLIENT\_AUTH3 | CK\_TRUST | Trusted for authenticating a client in a client/server module (example TLS/SSL/SSH) |
| CKA\_TRUST\_CODE\_SIGNING3 | CK\_TRUST | Trusted for authenticating a code fragment |
| CKA\_TRUST\_EMAIL\_PROTECTION3 | CK\_TRUST | Trusted for authenticating an email user. |
| CKA\_TRUST\_IPSEC\_IKE3 | CK\_TRUST | Trusted for IPSEC |
| CKA\_TRUST\_TIME\_STAMPING3 | CK\_TRUST | Trusted for Timestamping |
| CKA\_TRUST\_OCSP\_SIGNING3 | CK\_TRUST | Trusted for OCSP Signing |

1MUST be specified when the object is created. 2MUST be specified when the object is created unless all trust attributes are CKT\_TRUST\_UNKNOWN, or CKT\_NOT\_TRUSTED.

3 Missing CKA\_TRUST\_XXX attributes are treated as CKT\_TRUST\_UNKNOWN.

CKA\_TRUST\_XXX attributes map roughly to Certificate EKU values, and carry the same semantics. If CKA\_MODIFIABLE is not set in the template, the it defaults to CK\_TRUE, if CKA\_PRIVATE is not set in the template, it defaults to CK\_FALSE.

To obtain the effective trust attributes for a given certificate, a typical application will:

1. identify the tokens containing a [matching] trust object (with matching CKA\_HASH\_OF\_CERTIFICATE ???),
2. determine which of those tokens should be processed (presumably according to an established security policy), and
3. arrange those tokens in a list sorted in order of increasing priority.

An initial *working set* of attributes is obtained from the matching Trust object in the first (lowest priority) token in the list. The matching Trust objects of successive tokens are then iteratively merged into the working set as follows:

* if the value of a trust attribute in the current object is CKT\_TRUST\_UNKNOWN, the value of the corresponding trust attribute is left unchanged,
* otherwise the value of that trust attribute is replaced by the value in the current object.

The final effective set of trust attribute values are to be interpreted as follows:

|  |  |
| --- | --- |
| CKT\_TRUSTED | the certificate is trusted for the operation associated with the trust attribute |
| CKT\_TRUSTED\_DELEGATOR | for chain validation of ; t- |
| CKT\_NOT\_TRUSTED | tr; t  |
| CKT\_TRUST\_MUST\_VERIFY\_TRUST | thaving ; t |
| CKT\_TRUST\_UNKNOWN | T nor untrusted; tthe |

In the final set, CKT\_TRUST\_MUST\_VERIFY\_TRUST and CKT\_TRUST\_UNKNOWN have the same effect.

Note that when processing a certificate chain, applications may use the various Trust objects to override trust attribute values that would otherwise be associated with each certificate based solely on EKUs and other extensions encountered along the chain.

The following is a sample template for creating an X.509 certificate object:

CK\_OBJECT\_CLASS class = CKO\_CERTIFICATE;

CK\_UTF8CHAR label[] = “A certificate object”;

CK\_BYTE issuer[] = {...}; // *matches certificate’s issuer*

CK\_BYTE serialNumber[] = {...}; // matches certificate’s serialNumber

CK\_BYTE certificate[] = {...};

CK\_BBOOL true = CK\_TRUE;

CK\_TRUST trustedDelegator = CKT\_TRUSTED\_DELEGATOR;

CK\_TRUST notTrusted = CKT\_NOT\_TRUSTED;

CK\_MECHANISM\_TYPE hashMec = CKM\_SHA265

CK\_ATTRIBUTE template[] = {

 {CKA\_CLASS, &class, sizeof(class)},

 {CKA\_TOKEN, &true, sizeof(true)},

 {CKA\_LABEL, label, sizeof(label)-1},

 {CKA\_ISSUER, issuer, sizeof(issuer)},

 {CKA\_SERIAL\_NUBMER, serialNumber, sizeof(serialNumber)},

 {CKA\_HASH\_OF\_CERTIIFICATE, hash(hashnec,certificate, sizeof(certificate),hashLen(hashMec)},

 {CKA\_NAME\_HASH\_ALGORITHM, &hashMech, sizeof(hashMech)},

 {CKA\_TRUST\_SERVER\_AUTH, &trustedDelegator, sizeof(trustedDelegator) },

 {CKA\_TRUST\_OBJECT\_SIGNING, &notTrusted, sizeof(notTrusted) }

 // other attributes are CKT\_TRUST\_UNKNOWN if not included here.

};

### WTLS public key certificate objects

---------------------------------------

*NOTES: not part off the spec.*

*1. NSS currently has Vendor specific defines for CKO\_TRUST, CKA\_TRUST\_XXX and CKT\_TRUST\_XXX.*

*2. NSS has defines for CKA\_TRUST\_IPSEC\_END\_SYSTEM, CKA\_TRUST\_IPSEC\_USER, CKA\_TRUST\_IPSEC\_TUNNEL. These EKU’s are marked deprecated in the RFC’s and NSS never stored or red them.*

*3. NSS has defines for CKA\_TRUST\_TIME\_STAMPING, but hasn’t used them (will in the future).*

*4. NSS does not have a define for CKA\_TRUST\_OCSP\_SIGNING.*

*5. NSS has a define for CKA\_TRUST\_STEP\_UP\_APPROVED which is an obsolute trust value which says the CA can issue certificates that would connect using strong crypto for clients that can only do weak crypto. I’ve dropped it from this proposal since this no longer applies in the real world.*

*6. NSS has defined CKA\_CERT\_SHA1\_HASH and CKA\_CERT\_MD5\_HASH. I’ve replaced those in this proposal with a single hash and a hash mechanism models after the subject public key hash attribute.*

*7. NSS only accepts trust objects that don’t match the hash mechanism if the trust object has CKT\_TRUST\_UNKNOWN or CKT\_NOT\_TRUSTED for all trust attributes NSS processes.*

*8. NSS uses trust priority to override fixed builtin sources of trust which are not modifiable (read only tokens) with it’s database. This allows the fixed builtin sources to be changed on software update without perturbing the user’s own configuration.*

*9. On RHEL, ca-certificates export the NSS private trust objects using pk11-kit, which replaces the normal nss buildins module (libckbi.so). Ca-certificates uses pk11-kit to also build certlists consumable by openssl and other applications that don’t use trust objects.*