Secure Management of Objects in Remote Containers Shared by Multiple and Independent Issuers

Background to the Invention

- Issuing entities issue objects to object consumers
- An object consumer may be a consumer of objects from multiple issuers while keeping received objects in a single secure container
- Issuers are supposed to be the actual owner (manager) of the objects they issue, while object consumers typically only are "users" of such objects
- Issuers do in most cases not have direct access to the object consumers' secure containers, but rather depend on remote (on-line) object issuance procedures.

Put into real-world terms, issuing entities could be financial institutions, while object consumers could be individuals equipped with virtual credit-cards (objects) stored and executing inside of TPMs (Trusted Platform Modules).

The object consumer container could advantageously be an integral part of a mobile computer or mobile phone, giving a multitude of object issuance and usage possibilities.

Problem Description

If an issuer wishes to *replace*, *modify*, *clone*, or *delete* a *previously issued object* in a remote container *shared* by *multiple and independent issuers*, there are a number of requirements needing attention for making this viable:

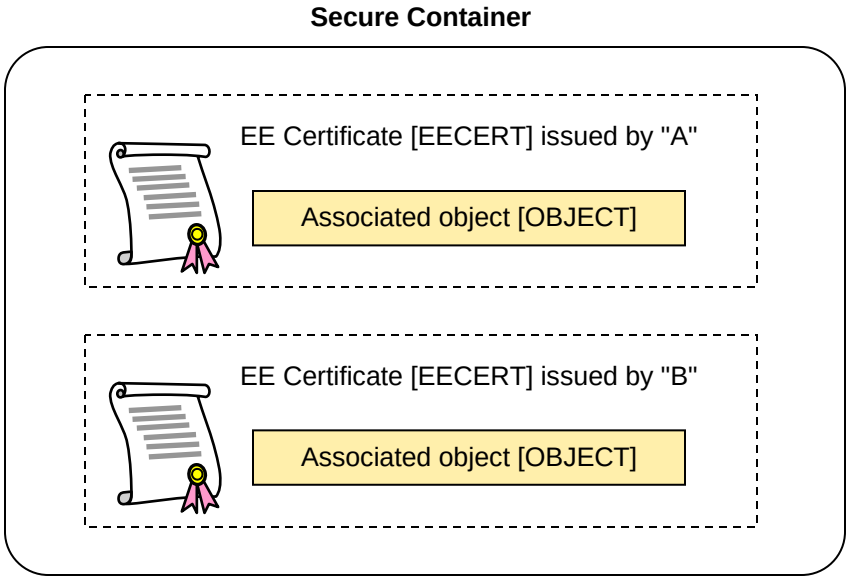
1. Aiding the object consumer in selecting the proper target object
2. Limiting an issuer's ability manipulating objects it does not own (is not the issuer of)

Proposed Solution

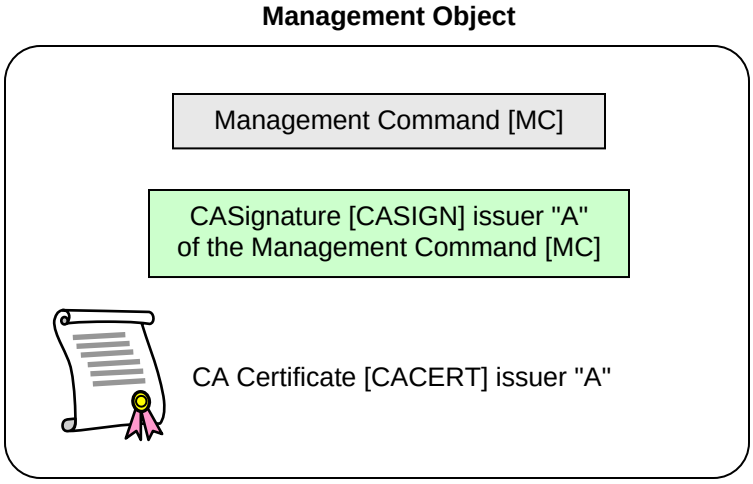
Notation: Items within brackets [] denote elements in the pictures featured in this specification.

Each issued object [OBJECT] is augmented by an X.509 EE (end-entity) certificate [EECERT] serving as a GUID (Globally Unique ID) with the added capability that it can be securely bound to a specific CA (issuer).

Below is an example of a pair of objects issued to an object consumer by issuer "A" and "B" respectively:



Now assume that issuer "A" wants to manage the object above which it has already issued. To do that the issuer applies the CA private key (not shown here) and the CA certificate [CACERT] it used when issuing the object's end-entity certificate [EECERT], but this time for signing [CASIGN] a Management Command [MC] according to the following:



The completed management object is subsequently sent to the object consumer for execution.

Due to the fact that a Management Command [MC] is uniquely bound to an issuer by the signature [CASIGN] and CA certificate [CACERT], the object consumer (software) system can automatically and securely restrict object management operations to only apply to objects with matching end-entity certificates [EECERT]. I.e., in addition to a valid CA signature [CASIGN], [CACERT] and the [EECERT] of the target [OBJECT] must form a valid two-element certificate-path in order to enable the execution of an object management operation.

Note that there is no need for object consumers "trusting" an issuer by having the issuer's root certificate installed. This is compliant with, for example, credit-cards which do not have to be verified by consumers, only by relying parties (merchants).

There are two basic variants covered by this invention disclosure:

1. Single-object references are typically specified by a SHA1 hash of the target end-entity certificate [EECERT]
2. Attribute references. Since X.509 end-entity certificates [EECERT] support a number of standardized attributes, a Management Command [MC] may specify object [OBJECT] selection attribute patterns like:

   Subject = "CN=John Doe,O=example.com,C=US"
   Email = "*@example.com"

Attribute references potentially facilitate management operations spanning multiple objects.

Object management operations may also be conditional (indicated by a flag or similar), or Unconditional at the issuer's discretion. That is, a Management Command [MC] may (for example) only be carried out if there already is a specific object [OBJECT] available.

Multiple Management Commands [MCs] may be aggregated, while still only requiring a single CA signature [CASIGN] and CA certificate [CACERT].

Implementation

The described scheme has been used for creating a universal "key-ring" supported by a universal key issuance and management protocol, where the end-entity certificates [EECERT] function as universal object-IDs for PKI keys, symmetric keys, PIN and PUK policy objects, key-attributes, logotypes, crypto-algorithm code, arbitrary extension data, as well as for meta-data schemes like Microsoft Information Cards.

Version: 0.18
Date: January 31, 2009

Author:

Anders Rundgren
Storbolsäng 50
74010 Almunge
Sweden

e-mail: anders.rundgren@telia.com

On the next couple of pages there is an example how this scheme has been utilized in the mentioned implementation known as KeyGen2.

The following XML fragment is an example of an object issuance message from an issuer to an object consumer. Note that this message is the final message in a KeyGen2 provisioning session. The depicted XML document actually contains two issued objects; one is just the EE-certificate [EECERT] itself which private key already is in a local storage, while the other object is a composite object holding a symmetric key, key attributes, and logotypes associated with the key.

```
<CredentialDeploymentRequest  ... >

  <CertifiedPublicKey ID="Key.1">
    <ds:X509Data>
      <ds:X509Certificate>MIIDXTCCAkWg … GVTVq7G0qJvQ==</ds:X509Certificate>
    </ds:X509Data>
  </CertifiedPublicKey>

  <CertifiedPublicKey FriendlyName="My OTP" ID="Key.2">
    <ds:X509Data>
      <ds:X509Certificate>MIICjDCCAXS … hwXnqZMGhO42s=</ds:X509Certificate>
    </ds:X509Data>
    <PiggybackedSymmetricKey EndorsedAlgorithms="http://www.w3.org/2000/09/xmldsig#hmac-sha1"
                                      MAC="82A6Ns3abk/3QCPQdwr9DOGV1b4=">
      <xenc:EncryptedKey>
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <xenc:CipherData>
          <xenc:CipherValue>FfrpTDmsAZKu9xR …  zzIKxrHiCPTsq/E=</xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedKey>
    </PiggybackedSymmetricKey>
    <PropertyBag Type="http://xmlns.webpki.org/keygen2/1.0#provider.ietf-hotp">
      <Property Name="LoginID" Value="johndoe"/>
      <Property Name="Counter" Value="0" Writable="true"/>
      <Property Name="Digits" Value="8"/>
    </PropertyBag>
    <Logotype MimeType="image/png"
              Type="http://xmlns.webpki.org/keygen2/1.0#logotype.application">iVBORw0KGgoA …
                                                                              0AAAAASUVORK5CYII=</Logotype>
    <Logotype MimeType="image/png"
              Type="http://xmlns.webpki.org/keygen2/1.0#logotype.card">iVBORw0KGgoA …
                                                    JbgkBJfgykv+BZ8Gg0jx8gdnAAAAAElFTkSuQmCC</Logotype>
  </CertifiedPublicKey>

</CredentialDeploymentRequest>
```

Note that although the second object is a symmetric key, an associated [EECERT] is still needed. Otherwise it cannot be managed by the scheme presented in this proposal. In Keygen2, the [EECERT] is also used for encryption of symmetric keys and other secret data associated with the [OBJECT].

The following Management Object holds a Management Command [MC] in the element named `DeleteKeysByContent` which will delete all objects having a valid [CACERT] [EECERT] certificate path where the [EECERT] also has an e-mail address attribute `john.doe@example.com`. `Signature` holds an enveloped Management Command [MS] CA-signature [CASIGN] while `X509Certificate` holds the associated CA-certificate [CACERT].

```
<KeyOperationRequest  ... >

   <ManageObject ID="Section.1">
      <DeleteKeysByContent Email="john.doe@example.com"/>
      <ds:Signature>
         <ds:SignedInfo>
            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <ds:Reference URI="#Section.1">
               <ds:Transforms>
                  <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                  <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
               </ds:Transforms>
               <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
               <ds:DigestValue>Imu7Pf88OjmLh+qNtwKzMwHgwrE=</ds:DigestValue>
            </ds:Reference>
         </ds:SignedInfo>
         <ds:SignatureValue>bIJU62/FPbg5P2 ... WLhumtkKH11BQ==</ds:SignatureValue>
         <ds:KeyInfo>
            <ds:X509Data>
               <ds:X509IssuerSerial>
                  <ds:X509IssuerName>CN=Example Root CA,O=Example,C=us</ds:X509IssuerName>
                  <ds:X509SerialNumber>2</ds:X509SerialNumber>
               </ds:X509IssuerSerial>
               <ds:X509Certificate>MIIDbzCCAlegA ... TZcbKHyCZvQ==</ds:X509Certificate>
            </ds:X509Data>
         </ds:KeyInfo>
      </ds:Signature>
   </ManageObject>

</KeyOperationRequest>
```