~~#### Out-of-band root CA Public Key~~
~~{: id="sect-5.2.5"}~~

Moved to Appendix

Root CA Key Update
~~{: id="sect-4.4"}~~ **{#sect-f}**

This discussion only applies to CAs that are directly trusted by some end entities. Self-signed CAs SHALL be considered as directly trusted CAs. Recognizing whether a non-self-signed CA is supposed to be directly trusted for some end entities is a matter of CA policy and is thus beyond the scope of this document.

~~The basis of the procedure described here is that the~~

RFC4210 describes extra certificates used by a CA ~~protects~~ **to protect** its new public ~~key using its previous private key~~ **key. This method has been shown to be very use case specific**
and ~~vice versa. Thus,~~
~~when a CA updates its key pair it must generate two extra cACertificate attribute values if certificates~~ **no assumptions** are ~~made available~~
~~using an X.500 directory (for a total of four: OldWithOld, OldWithNew, NewWithOld,~~ **done on this aspect** and ~~NewWithNew).~~
RootCaKeyUpdateContent is updated to specify the extra fields as optional.

When a CA changes its key pair, those entities who have acquired the old CA public key via "out-of-band" means are most affected. ~~It is these~~ **These** end
entities ~~who will~~ need ~~access~~ to **acquire** the new CA public key ~~protected with the old CA private key. However, they will only require this for~~ **in** a ~~limited period (until they have acquired the new CA public key via the "out-of-band" mechanism). This will typically be easily achieved when these end entities'~~ **trusted way.**
While noting that using link certificates ~~expire.~~

~~The data structure used to protect the new and old CA public keys is a standard certificate (which may also contain extensions). There are no new data structures required.~~

~~Note 1: This scheme does not make use of any of~~ **in** the ~~X.509 v3 extensions as it must be able to work even for version 1 certificates. The presence~~ **form** of ~~the KeyIdentifier extension would make~~ **NewWithOld is used** for ~~efficiency improvements.~~

~~Note 2:. While the scheme could be generalized to cover cases where the CA updates its key pair more than once during the validity period of one of its end entities' certificates,~~ this ~~generalization seems of dubious value. Not having~~ **in some use cases making assumption on** this ~~generalization simply means that the validity periods of certificates issued with the old CA key pair cannot exceed the end of the OldWithNew validity period.~~

~~Note 3: This scheme ensures that end entities will acquire the new CA public key, at the latest by~~ **aspect is beyond** the ~~expiry~~ **scope** of ~~the last certificate they owned that was signed with the old CA private key (via the~~

~~"out-of-band" means). Certificate and/or key update operations occurring at other times do not necessarily require~~ this ~~(depending on the end entity's equipment).~~

~~Note 4:~~ **document.**

In practice, a new root CA may have a slightly different subject DN, e.g., indicating a generation identifier like the year of issuance or a version number, for instance in an OU element. How to bridge trust to the new root CA certificate in a CA DN change or a cross-certificate scenario
is out of scope for this document.

~~###~~

CA Operator Actions
{: ~~id="sect-4.4.1"}~~ **id="sect-f.1"}**

To change the key of the CA, the CA operator does the following:

1. Generate a new key pair;

1. Create a certificate containing the ~~old~~ **new** CA public key signed with the new private key (the ~~"old~~ **"new** with new" certificate);

1. **Optionally:** Create a certificate containing the new CA public key signed with
the old private key (the "new with old" **or sometimes referred to as "link"** certificate);

1. **Optionally:** Create a certificate containing the ~~new~~ **old** CA public key signed with
the new private key (the ~~"new~~ **"old** with new" certificate);

1. Publish these new certificates ~~via the repository and/or other means (perhaps using a CAKeyUpdAnn message or RootCaKeyUpdateContent);~~

~~1. Export the new CA public key~~ so that end entities may acquire ~~it using the "out-of-band" mechanism (if required).~~ **it..**

The old CA private key is then no longer required. However, the old CA public key will remain in use for some time. The old CA public key is no longer required (other than for non-repudiation) when all end entities of this CA have securely acquired the new CA public key.

The ~~"old with new" certificate must have a validity period with the same notBefore and notAfter time as the "old with old" certificate.~~

~~The~~ "new with old" certificate must have a validity period with the same notBefore time as the "new with new" certificate and a notAfter time by which
all end entities of this CA will securely possess the new CA public key (at the latest, at the notAfter time of the "old with old" certificate).

The "new with new" certificate must have a validity period with a notBefore time that is before the notAfter time of the "old with old" certificate and a notAfter time that is after the notBefore time of the next update of this certificate.

The "old with new" certificate must have a validity period with the same notBefore and notAfter time as the "old with old" certificate.

Note: Further operational considerations on transition from one root CA self-signed certificate to the next is available in [RFC 8649 Section 5](#RFC8649).

~~### Verifying Certificates~~
~~{: id="sect-4.4.2"}~~

~~Normally when verifying a signature, the verifier verifies (among other things) the certificate containing the public key of the signer. However, once a CA is allowed to update its key there are a range of new possibilities. These are shown in the table below.~~

~~....~~

	~~Repository contains NEW and OLD public keys~~		~~Repository contains only OLD public key (due to, e.g., delay in publication)~~	
	~~PSE Contains NEW public key~~	~~PSE Contains OLD public key~~	~~PSE Contains NEW public key~~	~~PSE Contains OLD public key~~
~~Signer's certificate is protected using NEW key pair~~	~~Case 1: This is the standard case where the verifier can directly verify the certificate without using the repository~~	~~Case 3: In this case the verifier must access the repository in order to get the value of the NEW public key~~	~~Case 5: Although the CA operator has not updated the repository the verifier can verify the certificate directly - this is thus the same as case 1.~~	~~Case 7: In this case the CA operator has not updated the repository and so the verification will FAIL~~
~~Signer's certificate is protected using OLD key pair~~	~~Case 2: In this case the verifier must access the repository in order to get the value of the OLD public key~~	~~Case 4: In this case the verifier can directly verify the certificate without using the repository~~	~~Case 6: The verifier thinks this is the situation of case 2 and will access the repository; however, the verification will FAIL~~	~~Case 8: Although the CA operator has not updated the repository the verifier can verify the certificate directly - this is thus the same as case 4.~~

~~....~~

~~Note: Instead of using a repository, the end entity can use the root CA update general message to request the respective certificates from a PKI management entity, see {{sect-5.3.19.15}}, and follow the required validation steps.~~

~~#### Verification in Cases 1, 4, 5, and 8~~
~~{: id="sect-4.4.2.1"}~~

~~In these cases, the verifier has a local copy of the CA public key that can be used to verify the certificate directly. This is the same as the situation where no key change has occurred.~~

~~Note that case 8 may arise between the time when the CA operator has generated the new key pair and the time when the CA operator stores the updated attributes in the repository. Case 5 can only arise if the CA operator has issued both the signer's and verifier's certificates during this "gap" (the CA operator SHOULD avoid this as it leads to the failure cases described below)~~

~~#### Verification in Case 2~~
~~{: id="sect-4.4.2.2"}~~

~~In case 2, the verifier must get access to the old public key of the CA. The verifier does the following:~~

~~1. Look up the caCertificate attribute in the repository and pick the OldWithNew certificate (determined based on validity periods; note that the subject and issuer fields must match);~~

~~1. Verify that this is correct using the new CA key (which the verifier has locally);~~

~~1. If correct, check the signer's certificate using the old CA key.~~

~~Case 2 will arise when the CA operator has issued the signer's certificate, then changed the key, and then issued the verifier's certificate; so it is quite a typical case.~~

~~#### Verification in Case 3~~
~~{: id="sect-4.4.2.3"}~~

~~In case 3, the verifier must get access to the new public key of the CA. In case a repository is used, the verifier does the following:~~

~~1. Look up the cACertificate attribute in the repository and pick the NewWithOld certificate (determined based on validity periods; note that the subject and issuer fields must match);~~

~~1. Verify that this is correct using the old CA key (which the verifier has stored locally);~~

~~1. If correct, check the signer's certificate using the new CA key.~~

~~Case 3 will arise when the CA operator has issued the verifier's certificate, then changed the key, and then issued the signer's certificate; so it is also quite a typical case.~~

~~Note: Alternatively, the verifier can use the root CA update general message~~
~~to request the respective certificates from a PKI management entity, see {{sect-5.3.19.15}}, and follow the required validation steps.~~

~~#### Failure of Verification in Case 6~~
~~{: id="sect-4.4.2.4"}~~

~~In this case, the CA has issued the verifier's PSE, which contains the new key, without updating the repository attributes. This means that the verifier has no means to get a trustworthy version of the CA's old key and so verification fails.~~

~~Note that the failure is the CA operator's fault.~~

~~#### Failure of Verification in Case 7~~
~~{: id="sect-4.4.2.5"}~~

~~In this case, the CA has issued the signer's certificate protected with the new key without updating the repository attributes. This means that the verifier has no means to get a trustworthy version of the CA's new key and so verification fails.~~

~~Note that the failure is again the CA operator's fault.~~

~~### Revocation - Change of CA Key~~
~~{: id="sect-4.4.3"}~~

~~As we saw above, the verification of a certificate becomes more complex once the CA is allowed to change its key. This is also true for revocation checks as the CA may have signed the CRL using a newer private key than the one within the user's PSE.~~

~~The analysis of the alternatives is the same as for certificate verification.~~

~~###~~

Out-of-band root CA Public Key
{: ~~id="sect-5.2.5"}~~ **id="sect-f.2"}**

Each root CA must be able to publish its current public key via some "out-of-band" means. While such mechanisms are beyond the scope of this document, we define data structures that can support such mechanisms.

There are generally two methods available: either the CA directly publishes its self-signed certificate, or this information is available via the Directory (or equivalent) and the CA publishes a hash of this value to allow verification of its integrity before use.

~~~~ asn.1
  OOBCert ::= Certificate
~~~~

The fields within this certificate are restricted as follows:

* The certificate MUST be self-signed (i.e., the signature must be verifiable using the SubjectPublicKeyInfo field);

* The subject and issuer fields MUST be identical;

* If the subject field is NULL, then both subjectAltNames and issuerAltNames extensions MUST be present and have exactly the same value;

* The values of all other extensions must be suitable for a self-signed certificate (e.g., key identifiers for subject and issuer must be the same).

~~~~ asn.1
  OOBCertHash ::= SEQUENCE {
     hashAlg     [0] AlgorithmIdentifier     OPTIONAL,
~~~~

```
    certId      [1] CertId                  OPTIONAL,
    hashVal         BIT STRING
  }
~~~~
```

The intention of the hash value is that anyone who has securely received the hash value (via the out-of-band means) can verify a self-signed certificate for that CA.