RPKI Testing Data and Procedures

General Principles

Before providing the specifics of the data, we outline a few general principles of how we specify the data and procedures:

- Objects are denoted in the form $Xi_1i_2\ldots i_n$, where
  - X is a letter: C=cert, L=CRL, R=ROA, M=manifest
  - $i_1i_2\ldots i_{n-1}$ is the numerical identifier of the parent
  - i_n is a unique identifier among children of the given parent

  For example, C143 may be the parent for C1431, C1432, L1431, R1431, R1432, R1433 and M1431.
- Unless otherwise specified, it is assumed that the parent correctly signs an object. In the case when the signature is specified as incorrect, the object is signed by a certificate other than the parent.
- Unless otherwise specified, the validity times for objects (including next_update times) are specified so that the object is fully valid whenever the test is run. To pick arbitrary values, select the valid interval start time to be Jan. 1, 2000 and the end time to be Jan. 1, 2030. In the case when the object is specified to be expired or stale, pick the end time to be Jan. 1, 2001. If the object is premature, pick the start time to be Jan. 1, 2029.
- Forcing an object to expire or become stale is done by explicitly changing the database after the object has been loaded. For example, to make a certificate expire is done by the command:

  ```
  echo "update apki_cert set valto = \"2001-01-01 00:00:00\" where
        filename = \"blah.cer\";" | mysql $APKI_DB -u mysql
  ```

  The corresponding database tables for CRLs and manifests are apki_crl and apki_manifest respectively, and the field to set is next_upd. (Note: This way of handling expiration avoids time dependence for the test and allows us not to change the system clock.)
- All the data files can be placed statically in some test repository directory, and it actually does not matter whether the subdirectory structure mirrors the hierarchy. Specifying that particular objects/files are loaded into the database at a particular time is done by creating a rsync log file and feeding this as an argument into rsync_aur. Examples of how to do this are given in the trunk/testing directory.
- Relationships between objects, such as parent-child, are based on the values in the data fields of the objects. A certificate is a parent of a CRL or another certificate if the AKI of the child matches the SKI of the parent and the issuer of the child matches the subject of the parent. A certificate is the parent of a ROA if they both have the same SKI. Manifests do not directly have parents; instead, they have an embedded certificate that validates them and which has a parent. A CRL revokes a certificate if the certificate's serial number is in the CRL's serial number list and the two objects are siblings, i.e. share the same parent.
- The ROA filter query is the way that we determine (often indirectly) what the current state of the database is and whether the test results are as expected. The query is performed with the command line

  ```
  proto/query -a -s specsFile <-o outputFile>
  ```

The outputFile is optional and might better be replaced by "| tee outputFile". The specsFile tells whether to include ROAs that are in different states that are between valid and invalid. The format for the content of the specs file can be modeled on proto/sampleQuerySpecs, and tells whether each of the following cases should be allowed or not: StaleCRL, StaleManifest, StaleValidationChain, NoManifest. The tests will require multiple queries with different settings of the flags in the specs file.

- The ROA filter outputs should be designed to all be different for the different ROAs, and it is preferable that there be some easy way to match the filter output with the ROA in order to easily determine which ROAs were included in the output and which were not. (For example, the ROA's "number" could be included in the output.)
- If this is the first time running a particular procedure and the tester encounters a difference between the actual results and the expected results, there are three possibilities: (a) the expected results described in this document are not correct (and the document should be changed), (b) the data is incorrect or the procedure was not followed, and (c) there is an actual bug in code. Check out the first two possibilities before declaring a bug in code.

Data

- Certificates: C (trust anchor), C1, C2 (two copies with different filenames), C11, C12, C13, C21, C22, C23 (incorrectly signed), C111, C112, C113, C121, C131, C132, C211, C221, C231, C232, C1111, C2211, C2212
- CRLs: L11 (revokes C12 and C13), L111 (revokes C111), L112 (revokes C111 and C112), L121 (revokes C21 and C22)
- ROAs: R111, R221, R231, R1111, R1112, R1113, R1121, R1131, R1211, R1311, R1321, R1322, R2111, R2211, R2311, R2312, R2321, R11111, R22111, R22121, R22122 (incorrectly signed)
- Manifests:
  - M1 (contains C, C1 and C2)
  - M11 (contains C11, C12, C13, C22 – with a bad hash, C23)
  - M111 (contains C112, C113, C121, C131, C132, C211, C232 – with a bad hash, R111, R221, R231, L121)
  - M112 (contains same as M111 plus C221)
  - M231 (contains C111)
  - M1111 (contains C1111, C2211, R1111, R1112, R1113, R1121, R1131, R1211, R1311 – with a bad hash, R1321, R1322, R2111, R2211, R2311, R2312, R2321, R22111)

Test 1 - Validation and Trust Chains

This test focuses purely on the concept of trust chains and uses only certificates and ROAs. In particular, it tests the following cases:

- In-order trust chains, i.e. objects arriving in order of decreasing level of the hierarchy
- Out-of-order trust chains
- Invalidly signed certificates
- Missing links in trust chains
- Stale trust chains, i.e. formerly valid trust chains where a certificate has expired
- Formerly stale trust chains, i.e. an expired certificate replaced with a current one

The procedure is as follows:

1. Load all the certificates except C22 and the second copy of C2 in numerical order, followed by all the ROAs in any order, and finally certificate C22. (Note that the objects are loaded in the order that they are listed in the rsync log file.)
2. Run the query client with all four staleness filters set to "yes". Verify that all ROAs except R231, R2311, R2312, R2321, and R22122 are included in the filter file.
3. Repeat the query with the StaleValidationChain filter set to "no", and verify that the results are the same.
4. Change the expiration date for C2 so that it is now expired.
5. Run the garbage collector client.
6. Run the query client with all staleness filters set to "yes" and verify that nothing has changed from the previous query output.
7. Repeat the query with the StaleValidationChain filter set to "no", and verify that R2111, R2211, R11111 and R22111 are no longer included in the filter file.
8. Load the second copy of C2.
9. Repeat the query with the StaleValidationChain filter set to "no", and verify that R2111, R2211, R11111 and R22111 are once again included in the filter file.

Test 2 - CRLs and Revocations

This test focuses on how CRLs affect trust chains. In particular, it tests the following cases:

- A revoked certificate resulting in the lack of a trust chain, based on an in-order CRL, where in order means that the CRL is validated before the revoked certificate arrives
- A revoked certificate resulting in no trust chain, based on an out-of-order CRL
- A revoked certificate resulting in a stale trust chain
- A stale CRL
- A current CRL replacing a stale CRL

The procedure is as follows:

1. Load all the certificates and ROAs plus the CRLs L11, L111 and L121. The certificates should be in numerical order with L11 and L121 loaded between C12 and C13, and L111 loaded before everything. The ROAs should be loaded last.
2. Perform a query with all staleness filters set to "yes". Verify that the following ROAs (and no others) are part of the generated filters: R111, R221, R1121, R1131, R2111, R2211, R22111, and R22121.
3. Change the expiration date for L111 so that it is now expired.
4. Run the garbage collector client.
5. Perform the same query as before, and verify that the results are the same as before.
6. Perform a query with the StaleCRL filter set to "no". Verify that R1121 and R1131 are not part of the filter output.
7. Load the CRL L112, and run the garbage collector client.
8. Perform a query with the StaleCRL filter set to "no". Verify that R1121 is still not part of the output but R1131 is part of the output. Change the StaleCRL filter to "yes" and verify that this does not change the results.

Test 3 - Manifests

This test focuses on how manifests interact with other objects. In particular, it tests the following cases:

- Some objects not on manifests (while others are)
- A manifests without a trust chain
- In-order validated manifests, i.e. validated before the arrival of its referenced objects
- Out-of-order validated manifest
- In-order bad hash code, i.e. manifest validated before arrival of problem object
- Out-of-order bad hash code
- A stale manifest
- A previously stale manifest

The procedure is as follows:

1. Load all the certificates through the third level (up to and including C23), then load some of the ROAs, then load L121, then load all the manifests except M112, then load the remaining ROAs, and finally load the remaining certificates.
2. Perform a query with all staleness filters set to "yes". The missing ROAs should be: R221, R231, R1311, R2311, R2312, R2321, and R22122. The results should be the same if StaleManifest is set to "no".
3. Perform a query with the NoManifest staleness filter set to "no". The only ROAs that are present should be: R111, R1121, R1131, R1211, R1321, R1322, and R2111.
4. Change the expiration date for M111 so that it is now expired.
5. Run the garbage collector client.
6. Perform a query with both the NoManifest and StaleManifest filters set to "no". The only ROAs that are present should be: R111.
7. Load the manifest M112, and run the garbage collector client.
8. Perform a query with both the NoManifest and StaleManifest filters set to "no". The only ROAs that are present should be: R111, R1121, R1131, R1211, R1321, R1322, and R2111.