**Generating SCVP test program artifacts**

Step-by-step Guide

[This page intentionally left semi-blank]

Revision History

|  |  |
| --- | --- |
| Date | Notes |
| 2017-03-23 | First draft (no host name substitutions) |
| 2017-05-31 | Minor edits, added some additional appendices (F-H) |
| 2017-06-09 | Minor edits mostly addressing mods to PKITSv2 generation (distinct names and key ID suffixes per edition) |
| 2017-08-08 | Added reference to GSTPScriptRunner utility |

1 Overview 5

2 Test Artifacts 5

2.1 PKITS 5

2.1.1 Inputs 5

2.1.2 Generation Procedures 6

2.1.3 Outputs 8

2.2 PDTS 9

2.2.1 Inputs 9

2.2.2 Generation Procedures 9

2.2.3 Outputs 11

2.3 MFPKI 11

2.3.1 Inputs 11

2.3.2 Generation Procedures 11

2.3.3 Outputs 13

Appendix A - Python Virtual Environment creation 14

Appendix B - Resigning PKITS certificates with bad signatures 15

Appendix C - Using PITTv2 to review cloned artifacts 16

Appendix D - Test SCVP validation policy object identifiers 18

Appendix E – Re-rooting the MFPKI using existing test root CA 20

Appendix F – Updating PKITS to feature AIA and CRL DP 21

Appendix G – Sorting PKITS data into folders based on expected results 22

Appendix H – Tool inventory 23

# Overview

This document describes the generation of test artifacts that will be used to as part of GSA’s SCVP testing initiative. Three distinct sets of test artifacts will be employed:

1. NIST’s Public Key Infrastructure (PKI) Interoperability Test Suite (PKITS)
2. NIST’s Path Development Test Suite (PDTS)
3. Mock Federal PKI (MFPKI)

The remainder of this document describes the nature of each artifact set and procedures to generate them. Artifacts are generated using the PKI Copy and Paste (PCP) utility. For PKITS, PCP is used to generate test artifacts containing algorithms and key sizes other than RSA2048. For PDTS, PCP is used to refresh an expired test suite and to modify the URLs used for hosting. For MFPKI, PCP is used to generate artifacts of comparable complexity as the production Federal PKI. The following table describes the target end results.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Suite** | **Public Key Details** | **Hash Algorithm** | **Hosting Strategy** |
| PKITS | RSA 2048 | SHA256 | Not hosted; zip file |
| RSA 4096 | SHA256 | Not hosted; zip file |
| EC p256 | SHA256 | Not hosted; zip file |
| EC p384 | SHA384 | Not hosted; zip file |
| PDTS | RSA 2048 | SHA256 | Downloadable VM |
| RSA 2048 | SHA256 | CITE hosted |
| MFPKI | As observed  (mostly RSA 2048) | As observed (mostly SHA256) | Downloadable VM |

Note, the PKITS and PDTS varieties are not intended to be used simultaneously. Artifacts from one data set bear a strong resemblance to the corresponding artifacts in another data set. Each variety should be tested in isolation of the others.

# Test Artifacts

## PKITS

### Inputs

The PKITS\_data.zip file from <http://csrc.nist.gov/groups/ST/crypto_apps_infra/pki/pkitesting.html> will provide certificates and CRLs that will be input into PCP to facilitate cloning. Because DSA will not be used in the SCVP testing program, the following artifacts will not be cloned and can be omitted from the input data:

* DSACACert.crt
* DSAParametersInheritedCACert.crt
* InvalidDSASignatureTest6EE.crt
* ValidDSAParameterInheritanceTest5EE.crt
* ValidDSASignaturesTest4EE.crt
* DSACACRL.crl
* DSAParametersInheritedCACRL.crl

Additionally, since neither LDAP nor S/MIME is a target for the SCVP testing program, the certpairs and smime folders can be ignored entirely.

Two certificate objects must be resigned prior to cloning. These artifacts are: InvalidEESignatureTest3EE.crt and BadSignedCACert.crt. GoodCACert.p12 and TrustAnchorRootCertificate.p12 sign these artifacts, respectively. See Appendix B for steps to extract PKCS #8 keys from the PKCS #12 files and resign the two certificate files. This step need be performed just once, with the altered data set used as input to each cloning operation.

### Generation Procedures

#### Preparing PKITS for cloning

To prepare a PKITS data set for cloning, perform the following steps.

1. Download [PKITS\_data.zip](mailto:http://csrc.nist.gov/groups/ST/crypto_apps_infra/documents/PKITS_data.zip)[[1]](#footnote-1)
2. Extract zip file
3. Resign the necessary artifacts (InvalidEESignatureTest3EE.crt and BadSignedCACert.crt) using steps in Appendix B.
4. Use PkitsPdtsReduction utility to omit DSA artifacts
   1. python PkitsPdtsReduction.py -v <path to extracted zip>
5. Clean the CRLs folders used by PCP to store “real” CRLs and “fake” CRLs. The location is specified in **Options->Preferences->CRLs** folder. Delete the contents of the real and fake directories beneath the location identified in the CRL folder setting.
6. Optionally, delete log file (location specified in the dialog accessed via **Options->Preferences->LoggingConfiguration->Create/edit/view configuration**).
7. Create a new PCP database (**File->New PCP Database**)
8. Import PKITS certificates by navigating to the **Certificates** tab and clicking the **Import Certificates** button and browsing to the certs folder within the reduced PKITS\_data folder. 400 certificates should be imported.
9. Find the Invalid Missing basicConstraints EE Certificate Test 1 certificate (certificate hash value F5042289168F331674FCEE68D4170A0A640588D6) and delete it. Click **Import Certificate** and browse to the InvalidMissingbasicConstraintsTest1EE.crt to re-import it. This is necessary to establish the relationship to a certificate that was not imported as a CA.
10. Import PKITS CRLs by navigating to the **CRLs** tab and clicking the **Import CRLs** button and browsing to the crls folder within the reduced PKITS\_data folder. This will simply copy the files to the real folder beneath the configured CRL folder. 171 CRLs should be imported.
11. Save the PCP database.
12. Close the PCP database.

#### Basic PKITS clone generation

To prepare a cloned PKITS data set, perform the following steps:

1. Open the PCP database prepared per section 2.1.2.1 and altered via any customizations in 2.1.2.3-2.1.2.6.
2. Select the **Tools->Delete Fake PKI** and **Tools->Delete Fake Keys** to ensure new keys and artifacts will be generated. Select **Tools->Delete all fake items** if there are no custom configurations you wish to retain (do not choose this option is generating other than RSA 2048).
3. Make sure all options on the **Options->Preferences->Basic Generation Options** tab are unchecked. Otherwise, some negative test cases may not be accurately cloned.
4. Select the **Tools->Generate PKI** menu item to generate new key pairs and signed artifacts.
5. Wait (key generation will take some time).
6. Save the PCP database (possible via Save As to provide a name indicative of algorithm orientation of clones).
7. Review the **Has Fake** column on the Certificates and CRLs tabs and confirm all artifacts have been cloned. If not, review logs, determine cause, correct the issue and retry.
8. Select the **Tools->Export PKI** menu item.
9. Rename exported folder to indicate nature of cloned artifacts, if desired.
10. Copy contents of fake folder beneath configured CRL folder to the exported folder, if desired.
11. Generate a copy of the cloned artifacts using the original file names using the ClonedPkitsNameFixer tool along with original PKITS data, cloned data and a folder to receive renamed artifacts.
    1. mkdir <path>/PKITS\_<alg>/renamed
    2. python ClonedPkitsNameFixer.py -a /<path>/1.0.1/PKITS\_data -b /<path>/PKITS\_<alg> -c /<path>/PKITS\_<alg>/renamed
12. Break the signatures on necessary artifacts using the BreakSig utility
    1. python BreakSig.py -i /Users/cwallace/Desktop/SCVP\_artifacts/PKITS\_<alg>\_renamed/

#### Customizing generation rules for RSA 2048

No customization rules are required. Open the database prepared in section 2.1.2.1 then save a copy of the database using a name that indicates RSA2048 orientation. Next, simply execute the steps from 2.1.2.2.

#### Customizing generation rules for RSA 4096

Open the database prepared in section 2.1.2.1 then save a copy of the database using a name that indicates RSA4096 orientation.

Navigate to the Generator Configuration tab then to the Algorithm Map sub-tab. In the Mapped Algorithm column, choose “Algorithm rsaEncryption; Key size: 4096; Exponent: 0x010001” as the mapped value for “Algorithm rsaEncryption; Key size: 2048; Exponent: 0x010001”, which should be the only item in the Original Algorithm column.

Save the database then execute the steps in section 2.1.2.2. Note, key generation for RSA 4096 bit keys is extremely slow.

#### Customizing generation rules for ECDSA p256

Open the database prepared in section 2.1.2.1 then save a copy of the database using a name that indicates EC p256 orientation.

Navigate to the Generator Configuration tab then to the Algorithm Map sub-tab then Public key algorithms. In the Mapped Algorithm column, choose “Algorithm id\_ecPublicKey; Key size: 256; Curve: secp256r1” as the mapped value for “Algorithm rsaEncryption; Key size: 2048; Exponent: 0x010001”, which should be the only item in the Original Algorithm column. Change the Type to Digital signature algorithms and choose “ecdsa\_with\_SHA256” in the Mapped Algorithm column for the “Algorithm: sha256WithRSAEncryption; Parameters: present” option, which should be the only item in the Original Algorithm column.

Save the database then execute the steps in section 2.1.2.2.

#### Customizing generation rules for ECDSA p384

Open the database prepared in section 2.1.2.1 then save a copy of the database using a name that indicates EC p384 orientation.

Navigate to the Generator Configuration tab then to the Algorithm Map sub-tab. In the Mapped Algorithm column, choose “Algorithm id\_ecPublicKey; Key size: 384; Curve: secp384r1” as the mapped value for “Algorithm rsaEncryption; Key size: 2048; Exponent: 0x010001”, which should be the only item in the Original Algorithm column. Change the Type to Digital signature algorithms and choose “ecdsa\_with\_SHA384” in the Mapped Algorithm column for the “Algorithm: sha256WithRSAEncryption; Parameters: present” option, which should be the only item in the Original Algorithm column.

Save the database then execute the steps in section 2.1.2.2.

### Outputs

The result will include a complete set of PKITS artifacts with names that match the original filenames for each algorithm target elected. If each of sections 2.1.2.3-2.1.2.6 is executed, four sets of artifacts will result.

## PDTS

### Inputs

The PKITS\_data.zip file from <http://csrc.nist.gov/groups/ST/crypto_apps_infra/pki/pkitesting.html> will provide certificates and CRLs that will be input into PCP to facilitate harvesting then cloning. The bulk of PDTS is focused on LDAP. Since LDAP is not a target for the SCVP testring program, these artifacts need not be cloned. The following artifacts will be cloned (all other artifacts will be omitted):

* BasicHTTPURIPathDiscoveryOU1EE1.crt
* BasicHTTPURIPathDiscoveryOU1EE2.crt
* BasicHTTPURIPathDiscoveryOU1EE3.crt
* BasicHTTPURIPathDiscoveryOU1EE4.crt
* BasicHTTPURIPathDiscoveryOU1EE5.crt
* BasicHTTPURIPathDiscoveryOU3EE1.crt
* BasicHTTPURIPathDiscoveryOU3EE2.crt
* BasicHTTPURIPathDiscoveryOrg2EE1.crt
* BasicHTTPURIPathDiscoveryOrg2EE2.crt
* BasicHTTPURIPathDiscoveryOrg2EE3.crt
* BasicHTTPURIPathDiscoveryOrg2EE4.crt
* BasicHTTPURIPathDiscoveryOrg2EE5.crt
* BasicHTTPURIPathDiscoveryTest2EE.crt
* BasicHTTPURIPathDiscoveryTest4EE.crt
* RudimentaryHTTPURIPathDiscoveryTest13EE.crt
* RudimentaryHTTPURIPathDiscoveryTest14EE.crt
* RudimentaryHTTPURIPathDiscoveryTest15EE.crt
* RudimentaryHTTPURIPathDiscoveryTest16EE.crt
* RudimentaryHTTPURIPathDiscoveryTest2EE.crt
* RudimentaryHTTPURIPathDiscoveryTest4EE.crt
* RudimentaryHTTPURIPathDiscoveryTest7EE.crt
* RudimentaryHTTPURIPathDiscoveryTest8EE.crt
* BasicHTTPURITrustAnchorRootCert.crt

As with PKITS, since S/MIME is not a target for the SCVP testing program, the smime folder can be ignored entirely.

### Generation Procedures

#### Preparing PDTS for cloning

To prepare a PDTS data set for cloning, perform the following steps.

1. Download [PathDiscoveryTestSuite.zip](mailto:http://csrc.nist.gov/groups/ST/crypto_apps_infra/documents/PathDiscoveryTestSuite.zip)
2. Extract the zip file
3. Use PkitsPdtsReduction utility to omit LDAP artifacts
   1. python PkitsPdtsReduction.py -d <path to extracted zip>
4. Clean the CRLs folders used by PCP to store “real” CRLs and “fake” CRLs. The location is specified in **Options->Preferences->CRLs** folder. Delete the contents of the real and fake directories beneath the location identified in the CRL folder setting.
5. Optionally, delete log file (location specified in the dialog accessed via **Options->Preferences->LoggingConfiguration->Create/edit/view configuration**).
6. Create a new PCP database (**File->New PCP Database**)
7. Import PDTS end entity certificates by navigating to the **Certificates** tab and clicking the **Import Certificates** button and browsing to the “End Entity Certs” folder within the reduced “Path Discovery Test Suite” folder. 22 certificates should be imported.
8. Import PDTS trust anchor certificates by navigating to the **Certificates** tab and clicking the **Import Certificates** button and browsing to the “Trust Anchor Certs” folder within the reduced “Path Discovery Test Suite” folder. 1 additional certificate should be imported resulting in 23 certificates total.
9. Save the PCP database.
10. Make sure both **Recursive URI harvest** and **Skip LDAP URIs during** harvest are checked. Harvest additional certificates by clicking the **Harvest certificates from URIs** button on the **Certificates** tab. 82 certificates should be present, along with 42 PKCS #7 messages.
11. Navigate to the CRLs tab. Make sure **Skip LDAP URIs during harvest** is checked the click **Harvest CRLs** to harvest CRLs. 28 CRLs should be retrieved.
12. Save the PCP database.
13. Close the PCP database.

#### Basic PDTS clone generation

To prepare a cloned PDTS data set, perform the following steps:

1. Open the PCP database prepared in section 2.2.2.1.
2. Select the **Tools->Delete Fake PKI** and **Tools->Delete Fake Keys** to ensure new keys and artifacts will be generated. Select **Tools->Delete all fake items** if there are no custom configurations you wish to retain.
3. Navigate to the **Generator Configuration** tab. On the **Hosts** sub-tab select the **URI** name form. Click the **Append default suffix to each** button. Enter test into the resulting dialog and click OK. The names from the left column should now appear in the right column with a .test suffix appending. There is no need to alter the RFC822 domain and are no other hosts listed for other name forms.
4. Make sure the first two options on the **Options->Preferences->Basic Generation Options** tab are checked. This will ensure expired certificates and stale CRLs are refreshed. This is a necessary step because PDTS was never updated by NIST after the initial data set expired.
5. Select the **Tools->Generate PKI** menu item to generate new key pairs and signed artifacts.
6. Wait (key generation will take some time).
7. Save the PCP database (possible via Save As to provide a name indicative of algorithm orientation of clones).
8. Review the **Has Fake** column on the Certificates and CRLs tabs and confirm all artifacts have been cloned. If not, review logs, determine cause, correct the issue and retry.
9. Select the **Tools->Export PKI** menu item.
10. Rename exported folder to indicate nature of cloned artifacts, if desired.
11. Copy contents of fake folder beneath configured CRL folder to the exported folder, if desired.
12. Generate a copy of the cloned artifacts using the original file names using the ClonedPkitsNameFixer tool along with original PDTS data, cloned data and a folder to receive renamed artifacts.
    1. mkdir <path>/PDTS/renamed
    2. python ClonedPkitsNameFixer.py -d “/<path>/Path Discovery Test Suite” -e /<path>/PDTS -f /<path>/PDTS/renamed
    3. Delete spurious folders created with names of PKITS path settings
13. Export and save a list of hosts using the **Analysis->Reports->List hosts** menu item.

### Outputs

The result will include a complete set of PDTS artifacts with names for end entity and trust anchor certificates that match the original filenames. These materials can be used to prepare a VM hosting the artifacts.

## MFPKI

### Inputs

A set of 58 end entity certificates from various PKIs connected to the FPKI will be provided as input to PCP along with a p7b file containing all certificates collected by the FPKI crawler (available from <https://fpki-graph.fpki-lab.gov/crawler/>) to facilitate harvesting then cloning.

### Generation Procedures

#### Preparing MFPKI for cloning

To prepare a MFPKI data set for cloning, perform the following steps.

1. Collect the desired end entity certificates beneath a single folder (there may be sub-folders) and download the latest FPKI crawler [p7b file](mailto:https://fpki-graph.fpki-lab.gov/crawler/allCerts/paths/CACertificatesValidatingToFederalCommonPolicy.p7b).
2. Clean the CRLs folders used by PCP to store “real” CRLs and “fake” CRLs. The location is specified in **Options->Preferences->CRLs** folder. Delete the contents of the real and fake directories beneath the location identified in the CRL folder setting.
3. Optionally, delete log file (location specified in the dialog accessed via **Options->Preferences->Logging Configuration->Create/edit/view configuration**).
4. Create a new PCP database (**File->New PCP Database**)
5. Import MFPKI end entity certificates by navigating to the **Certificates** tab and clicking the **Import Certificates** button and browsing to the folder containing the end entity certificates collected in step 1. Confirm the number of certificates that were imported matches expectations.
6. Import the CA certificates from the FPKI crawler by navigating to the PKCS7 Messages tab, clicking the **Import PKCS7 File…** button and browsing to the CACertificatesValidatingToFederalCommonPolicy.p7b file.
7. Save the PCP database.
8. Make sure both **Recursive URI harvest** and **Skip LDAP URIs during** harvest are checked. Harvest additional certificates by clicking the **Harvest certificates from URIs** button on the **Certificates** tab. After that completes, click the **Harvest OCSP responder certificates** button.
9. Navigate to the CRLs tab. Make sure **Skip LDAP URIs during harvest** is checked the click **Harvest CRLs** to harvest CRLs.
10. Save the PCP database.
11. Close the PCP database.

#### Basic MFPKI clone generation

To prepare a cloned MFPKI data set, perform the following steps:

1. Open the PCP database prepared in section 2.2.2.1.
2. Select the **Tools->Delete Fake PKI** and **Tools->Delete Fake Keys** to ensure new keys and artifacts will be generated. Select **Tools->Delete all fake items** if there are no custom configurations you wish to retain.
3. Navigate to the **Generator Configuration** tab. On the **Hosts** sub-tab select the **URI** name form. Click the **Append default suffix to each** button. Enter test into the resulting dialog and click OK. The names from the left column should now appear in the right column with a .test suffix appending. There is no need to alter the RFC822 or other name forms since testing these name forms is not planned and these name forms have no hosting component.
4. Navigate to the **Basic Generator Configuration** sub-tab. Make sure cn=default is selected as the Configuration Name then click the **Generate configuration for selected configuration** option. Click all four check boxes associated with alterations to cause end entity personal information to be altered.
5. Make sure the first two options on the **Options->Preferences->Basic Generation Options** tab are checked. This will ensure expired certificates and stale CRLs are refreshed.
6. Navigate to the DN Map sub-tab. Go through the list and for each top level RDN (i.e., c=US, c=CA, dc=com, etc.) modify the name to indicate a test certificate by adding either o=Mock or dc=Mock adjacent to the terminal RDN.
7. Select the **Tools->Generate PKI** menu item to generate new key pairs and signed artifacts.
8. Wait (key generation will take some time).
9. Save the PCP database (possible via Save As to provide a name indicative of algorithm orientation of clones).
10. Review the **Has Fake** column on the Certificates and CRLs tabs and confirm all artifacts have been cloned. If not, review logs, determine cause, correct the issue and retry.
11. Select the **Tools->Export PKI** menu item.
12. Rename exported folder to indicate nature of cloned artifacts, if desired.
13. Copy contents of fake folder beneath configured CRL folder to the exported folder, if desired.
14. Export and save a list of hosts using the **Analysis->Reports->List hosts** menu item.

### Outputs

The result will include a complete set of PDTS artifacts with names for end entity and trust anchor certificates that match the original filenames. These materials can be used to prepare a VM hosting the artifacts.

# Appendix A - Python Virtual Environment creation

Several of the tools used to prepare the SCVP test artifacts are written in Python. The tools have minimal dependencies and can be run in a relatively bare bones Python 3 virtual environment. The following steps was used on an OS X system with Python 3 installed in /usr/local/bin.

/usr/local/bin/virtualenv pcpvenv --python=python3

The following step can be used to activate the virtual environment.

source pcpvenv/bin/activate

Install the glob2 package using the following command.

pip install glob2

# Appendix B - Resigning PKITS certificates with bad signatures

PCP uses signatures to organize artifacts for cloning. In order for certificates with bad signatures to be successfully cloned, the artifacts must first be resigned. This will enable PCP to generate a clone. Signatures can be broken on the cloned artifacts using the BreakSig.py script.

Two certificates require resigning, which requires extracting private keys for two different CAs. The following examples illustrate how to extract PKCS #8 keys from the PKCS #12 objects included with PKITS then using ResignCert to generate resigned artifacts suitable for cloning.

|  |
| --- |
| openssl pkcs12 -in TrustAnchorRootCertificate.p12 -nodes  -out TrustAnchorRootCertificate.pem |
| openssl pkcs8 -topk8 -inform PEM -outform DER  -in TrustAnchorRootCertificate.pem  -out TrustAnchorRootCertificate.p8 -nocrypt |
| openssl pkcs12 -in GoodCACert.p12 -nodes  -out GoodCACert.pem |
| openssl pkcs8 -topk8 -inform PEM -outform DER  -in GoodCACert.pem -out GoodCACert.p8 -nocrypt |
|  |
|  |
| ResignCert.exe -p TrustAnchorRootCertificate.p8  -i BadSignedCACert.crt -o .\BadSignedCACert.resigned.crt |
| ResignCert.exe -p GoodCACert.p8  -i InvalidEESignatureTest3EE.crt  -o .\InvalidEESignatureTest3EE.resigned.crt |

After generating resigned artifacts, rename thee original files with a .omit file extension.

# Appendix C - Using PITTv2 to review cloned artifacts

The PKI Interoperability Test Tool version 2 (PITTv2) can be used to verify cloned artifacts. This section describes how to use the tool with the PKITS.sdb file provided as a sample. This .sdb file contains certification path validation settings that align with the settings defined in the PKITS documentation. The settings are defined in terms of several artifacts that are assumed to exist. The table below describes these artifacts. The trust anchor store files can all co-exist. The contents of the PKITS\_data folder will need to be manually changed depending on the collection being verified. In other words, there is a security environment defined for each PKITS data set but there is only one set of path settings definitions that is shared across PKITS data sets. One could endeavor to define path settings definitions for each data set if desired.

|  |  |
| --- | --- |
| **File or folder location** | **Contents** |
| C:\PittSettings\tas\PKITS\_RSA\_2048.tas | Trust anchor store file containing the TrustAnchorRootCertificate.crt file from the cloned PKITS RSA 2048 collection. |
| C:\PittSettings\tas\PKITS\_RSA\_4096.tas | Trust anchor store file containing the TrustAnchorRootCertificate.crt file from the cloned PKITS RSA 4096 collection. |
| C:\PittSettings\tas\PKITS\_EC\_p256.tas | Trust anchor store file containing the TrustAnchorRootCertificate.crt file from the cloned PKITS EC p256 collection. |
| C:\PittSettings\tas\PKITS\_EC\_p384.tas | Trust anchor store file containing the TrustAnchorRootCertificate.crt file from the cloned PKITS EC p384 collection. |
| C:\PKITS\_data\certificates | CA certificates that align with the target collection being validated |
| C:\PKITS\_data\crls | CRLs certificates that align with the target collection being validated |

There are two test cases where the PITTv2 result does not match the “expected” result. For test case 4.14.16, PITTv2 does not show an error for a certificate that is on hold. However, it does return this information in the results (i.e., view the path log for this target). For test case 4.14.30, PITTv2 returns an error where the PKITS documentation indicates success should be returned. In this test case, the CRL issuer’s revocation status is determined using a CRL issued by the CRL issuer. PITTv2 does not allow circular dependencies (i.e., a CA may not vouch for itself). With these two caveats in mind, the following table describes the summary results generated by PITTv2 against the PKITS data set.

|  |  |  |  |
| --- | --- | --- | --- |
| **Settings** | **# of paths** | **# of certificates** | **# of valid paths** |
| Default settings | 204 | 210 | 92 |
| Settings 1 | 4 | 4 | 2 |
| Settings 2 | 1 | 1 | 1 |
| Settings 3 | 1 | 1 | 0 |
| Settings 4 | 2 | 2 | 1 |
| Settings 5 | 12 | 12 | 10 |
| Settings 6 | 8 | 8 | 5 |
| Settings 7 | 1 | 1 | 1 |
| Settings 8 | 2 | 2 | 0 |
| Settings 9 | 2 | 2 | 0 |
| Settings 10 | 1 | 1 | 0 |

If an artifact set is regenerated, the .tas file for that dataset must be updated to include the new trust anchor and to not include the old trust anchor.

# Appendix D - Test SCVP validation policy object identifiers

Given the various PKITS data sets are not intended to be comingled, a distinct set of validation policy object identifiers has been defined for each data set for use on SCVP servers configured for testing.

The validation policies are defined related to Red Hound Software’s OID arc: 1.3.6.1.4.1.37623. An OID has been defined for the SCVP program. Test suites are defined beneath the SCVP program OID with validation policy OIDs beneath the test suite OID.

-- 1.3.6.1.4.1.37623.10

id-scvp-testing OBJECT IDENTIFIER ::= { id-redhound 10 }

-- 1.3.6.1.4.1.37623.10.1

id-scvp-valpol OBJECT IDENTIFIER ::= { id-scvp-testing 1 }

-- 1.3.6.1.4.1.37623.10.1.1-4

id-scvp-pkits-2048 OBJECT IDENTIFIER ::= { id-scvp-valpol 1 }

id-scvp-pkits-4096 OBJECT IDENTIFIER ::= { id-scvp-valpol 2 }

id-scvp-pkits-p256 OBJECT IDENTIFIER ::= { id-scvp-valpol 3 }

id-scvp-pkits-p384 OBJECT IDENTIFIER ::= { id-scvp-valpol 4 }

id-scvp-pdts OBJECT IDENTIFIER ::= { id-scvp-valpol 5 }

id-scvp-mfpki OBJECT IDENTIFIER ::= { id-scvp-valpol 6 }

-- 1.3.6.1.4.1.37623.10.1.1.0-10

id-scvp-pkits-2048-def OBJECT IDENTIFIER ::= { id-scvp-pkits-2048 0 }

id-scvp-pkits-2048-1 OBJECT IDENTIFIER ::= { id-scvp-pkits-2048 1 }

id-scvp-pkits-2048-2 OBJECT IDENTIFIER ::= { id-scvp-pkits-2048 2 }

id-scvp-pkits-2048-3 OBJECT IDENTIFIER ::= { id-scvp-pkits-2048 3 }

id-scvp-pkits-2048-4 OBJECT IDENTIFIER ::= { id-scvp-pkits-2048 4 }

id-scvp-pkits-2048-5 OBJECT IDENTIFIER ::= { id-scvp-pkits-2048 5 }

id-scvp-pkits-2048-6 OBJECT IDENTIFIER ::= { id-scvp-pkits-2048 6 }

id-scvp-pkits-2048-7 OBJECT IDENTIFIER ::= { id-scvp-pkits-2048 7 }

id-scvp-pkits-2048-8 OBJECT IDENTIFIER ::= { id-scvp-pkits-2048 8 }

id-scvp-pkits-2048-9 OBJECT IDENTIFIER ::= { id-scvp-pkits-2048 9 }

id-scvp-pkits-2048-10 OBJECT IDENTIFIER ::= { id-scvp-pkits-2048 10 }

-- 1.3.6.1.4.1.37623.10.1.2.0-10

id-scvp-pkits-4096-def OBJECT IDENTIFIER ::= { id-scvp-pkits-4096 0 }

id-scvp-pkits-4096-1 OBJECT IDENTIFIER ::= { id-scvp-pkits-4096 1 }

id-scvp-pkits-4096-2 OBJECT IDENTIFIER ::= { id-scvp-pkits-4096 2 }

id-scvp-pkits-4096-3 OBJECT IDENTIFIER ::= { id-scvp-pkits-4096 3 }

id-scvp-pkits-4096-4 OBJECT IDENTIFIER ::= { id-scvp-pkits-4096 4 }

id-scvp-pkits-4096-5 OBJECT IDENTIFIER ::= { id-scvp-pkits-4096 5 }

id-scvp-pkits-4096-6 OBJECT IDENTIFIER ::= { id-scvp-pkits-4096 6 }

id-scvp-pkits-4096-7 OBJECT IDENTIFIER ::= { id-scvp-pkits-4096 7 }

id-scvp-pkits-4096-8 OBJECT IDENTIFIER ::= { id-scvp-pkits-4096 8 }

id-scvp-pkits-4096-9 OBJECT IDENTIFIER ::= { id-scvp-pkits-4096 9 }

id-scvp-pkits-4096-10 OBJECT IDENTIFIER ::= { id-scvp-pkits-4096 10 }

-- 1.3.6.1.4.1.37623.10.1.3.0-10

id-scvp-pkits-p256-def OBJECT IDENTIFIER ::= { id-scvp-pkits-p256 0 }

id-scvp-pkits-p256-1 OBJECT IDENTIFIER ::= { id-scvp-pkits-p256 1 }

id-scvp-pkits-p256-2 OBJECT IDENTIFIER ::= { id-scvp-pkits-p256 2 }

id-scvp-pkits-p256-3 OBJECT IDENTIFIER ::= { id-scvp-pkits-p256 3 }

id-scvp-pkits-p256-4 OBJECT IDENTIFIER ::= { id-scvp-pkits-p256 4 }

id-scvp-pkits-p256-5 OBJECT IDENTIFIER ::= { id-scvp-pkits-p256 5 }

id-scvp-pkits-p256-6 OBJECT IDENTIFIER ::= { id-scvp-pkits-p256 6 }

id-scvp-pkits-p256-7 OBJECT IDENTIFIER ::= { id-scvp-pkits-p256 7 }

id-scvp-pkits-p256-8 OBJECT IDENTIFIER ::= { id-scvp-pkits-p256 8 }

id-scvp-pkits-p256-9 OBJECT IDENTIFIER ::= { id-scvp-pkits-p256 9 }

id-scvp-pkits-p256-10 OBJECT IDENTIFIER ::= { id-scvp-pkits-p256 10 }

-- 1.3.6.1.4.1.37623.10.1.4.0-10

id-scvp-pkits-p384-def OBJECT IDENTIFIER ::= { id-scvp-pkits-p384 0 }

id-scvp-pkits-p384-1 OBJECT IDENTIFIER ::= { id-scvp-pkits-p384 1 }

id-scvp-pkits-p384-2 OBJECT IDENTIFIER ::= { id-scvp-pkits-p384 2 }

id-scvp-pkits-p384-3 OBJECT IDENTIFIER ::= { id-scvp-pkits-p384 3 }

id-scvp-pkits-p384-4 OBJECT IDENTIFIER ::= { id-scvp-pkits-p384 4 }

id-scvp-pkits-p384-5 OBJECT IDENTIFIER ::= { id-scvp-pkits-p384 5 }

id-scvp-pkits-p384-6 OBJECT IDENTIFIER ::= { id-scvp-pkits-p384 6 }

id-scvp-pkits-p384-7 OBJECT IDENTIFIER ::= { id-scvp-pkits-p384 7 }

id-scvp-pkits-p384-8 OBJECT IDENTIFIER ::= { id-scvp-pkits-p384 8 }

id-scvp-pkits-p384-9 OBJECT IDENTIFIER ::= { id-scvp-pkits-p384 9 }

id-scvp-pkits-p384-10 OBJECT IDENTIFIER ::= { id-scvp-pkits-p384 10 }

-- 1.3.6.1.4.1.37623.10.1.5.0

id-scvp-pdts-def OBJECT IDENTIFIER ::= { id-scvp-pdts 0 }

-- 1.3.6.1.4.1.37623.10.1.6.0

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 0 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 1 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 2 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 6 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 7 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 8 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 13 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 14 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 15 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 16 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 17 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 18 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 19 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 20 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 36 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 39 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 40 }

id-scvp-mfpki-def OBJECT IDENTIFIER ::= { id-scvp-mfpki 41 }

# Appendix E – Re-rooting the MFPKI using existing test root CA

Before the MFPKI was available, test PKIs were generated using CA products and some combination of manual and automated artifact generation procedures. The trust anchors associated with these efforts are in wide enough use, that tying the MFPKI to the existing trust anchors is desirable. This appendix describes steps to identity certificates signed by the cloned Federal Common Policy CA and cloned Federal Bridge CA 2016 so PKCS #10 requests can be generated using XCA to facilitate certificate issuance using the existing CA products.

<TODO>

# Appendix F – Updating PKITS to feature AIA and CRL DP

NIST’s PKITS test data requires manual presentation of certificates and CRLs to the path validation client. When testing some SCVP servers, the level of effort necessary to manually provision hundreds of certificates and CRLs is quite high. To avoid expending effort on a per-server basis during testing, the PKITS data was recut to feature authorityInformationAccess (AIA) and crlDistributionPoints (CRL DP) extensions to enable responders to automatically retrieve the information. The updated data set can be used as an alternative to the NIST data in section 2.1.2.1.

To prepare the updated data, several new scripts and tools were developed:

* AddAiaAndCrlDp is a C/C++ command line utility that adds AIAs to certificates and CRLs and CRL DPs to certificates. The PkitsUpdater.py script is used to drive the tool.
* FetchKeyId is a C/C++ command line utility that returns an ASCII hexadecimal representation of a SKID or AKID extension in a certificate or CRL.
* PkitsUpdater.py is a Python script that accepts a copy of the NIST PKITS edition and emits a data set containing certificates with appropriate AIAs and CRL DPs, CRLs with appropriate AIAs and a collection of PKCS7 files for hosting as AIA data.

Several additional existing tools were used as well including the ResignCert and ResignCrl utilities built for DISA and the openssl command line utility.

# Appendix G – Sorting PKITS data into folders based on expected results

The test client implements support for the lightweight, long-term and batch profiles defined in the “Treasury Validation Services: SCVP Request and Response Profile” document. The batch option requires support for processing requests containing up to 256 certificates. The MFPKI data set will be used for testing the upper boundary condition (because PKITS and PDTS lack sufficient numbers of end entity certificates). However, the MFPKI data set is intended to consist solely of valid certificates. To test processing a mixture of valid and invalid certificates, the PKITS data set is chunked into folders that indicate the expected results.

PKITS features 11 different path validation input possibilities. To facilitate exercising batch under different input scenarios, the certificates used within each possibility are subdivided into a folder indicating success is expected and a folder indicating failure is expected. The PkitsBatchOrganizer.py script is used to chunk data into appropriate folders suitable for use as inputs to the test client during batch testing.

# Appendix H – Tool inventory

This section describes each of the tools used to produce the data for the test program and for use during testing of products. The summary list is as follows:

|  |  |  |
| --- | --- | --- |
| **Tool** | **Source** | **Purpose** |
| PKI Copy and Paste (PCP) | Developed by Red Hound for GSA | Used to generate the PKITS, PDTS and MFPKI data sets |
| PKI Interoperability Test Tool v2 (PITTv2) | Developed by Red Hound for DISA | Used to test the PKITS, PDTS and MFPKI data sets |
| Trust Anchor Store Manager | Developed by Red Hound for DISA | Used to prepare trust anchor stores used by PITTv2 |
| OpenSSL | OpenSSL Software Foundation | Used by custom Python scripts for various purposes |
| ResignCert | Developed by Red Hound for DISA | Used by custom Python scripts to resign certificates |
| ResignCrl | Developed by Red Hound for DISA | Used by custom Python scripts to resign CRLs |
| xca | Christian Hohnstadt (from xca.sourceforge.net) | Optionally used to re-root the MFPKI to use an existing test trust anchor |

|  |  |  |
| --- | --- | --- |
| **Tool** | **Language** | **Purpose** |
| AddAiaAndCrlDp | C/C++ | Used to inject AIA and CRL DP extensions into certificates and AIA extensions into CRLs |
| BreakSig | Python | Used to alter signatures on PKITS files associated with bad signature test cases |
| ClonedPkitsNameFixer | Python | Used to rename cloned artifacts using name from NIST’s PKITS edition |
| FetchKeyId | C/C++ | Used to read SKID and AKID extensions in certificates and CRLs |
| GSTPScriptRunner | Python | Used to run scripts emitted by ScvpScriptGenerator with log rotation |
| PkitsBatchOrganizer | Python | Used to chunk PKITS data into folders based on expected results for a set of path validation inputs |
| PkitsPdtsReduction | Python | Used to rename files that will not be cloned (i.e., LDAP PDTS, DSA PKITS) |
| PkitsSorter | Python | Used to sort PKITS data into folders named with the path validation inputs that are used when validating the certificates |
| PkitsTableGenerator | C/C++ | Used to generate a CSV file with PKITS test cases and expected results enumerated |
| PkitsUpdater | Python | Used to generate PKITSv2 data set (i.e., PKITS with AIA and CRL DP extensions) |
| ScvpScriptGenerator | C/C++ | Used to generate bash scripts that can be used to drive the test client |
| vss2.jar | Java | Used to test SCVP responders (i.e., Java client based on Treasury’s SCVP code) |

1. Steps 1-3 apply when using NIST’s PKITS edition. See Appendix F for details on PKITSv2 data set (i.e., PKITS w/ AIA and CRL DP extensions). [↑](#footnote-ref-1)