Source Code Review:

Domain Name Security Extensions Key Signing Key Management Tools

version 1.01, April 2021

Reviewers:

Paul Wouters
Martin Ringehahn

Introduction

No Hats Corporation has performed a Source Code Review of the Domain Name Security Extensions ("DNSSEC") Key Signing Key Management Tools software to certify that:

A. There is a documented architectural design describing the security domains and functions maintained by the signer.
B. The architectural design demonstrates that the signer system prevents bypass of the security-enforcing functionality.
C. There is a functional specification completely representing the signer system and all operations associated with it.
D. There is a modular design description and a one-to-one correspondence with the modular decomposition of the implementation.
E. The implementation representation completely and accurately implements the security-enforcing functions.

The software version ("Software") that No Hats has reviewed can be identified by the following repository values:

ssh://github.com/iana-internal/dnssec-keytools-2019-dev.git

commit efd2237dd48e58cd0a621d0b0382b5217002a3ac
Merge: 07cad20 8c9ee14
Author: James Mitchell <james.mitchell@iana.org>
Date: Fri Apr 9 11:19:32 2021 -0700

Merge pull request #28 from iana-internal/doc-update

No Hats and all of its personnel that performed work on this contract have not been, directly or indirectly, involved in the creation of the Software.

No Hats has reviewed all the source code, documentation and other files that came with the Software. The complete list of engineers who worked on this audit are:

Paul Wouters paul@nohats.ca
Martin Ringehahn martin@nohats.ca

The audit was performed against the 5th edition:
https://www.iana.org/dnssec/procedures/ksk-operator/ksk-dps-20200407.txt

As a result of this audit, the DNSSEC Practise Statement was updated to the 6th edition:
https://www.iana.org/dnssec/procedures/ksk-operator/ksk-dps-20201104.html

Executive Summary

The software and the unit tests have been audited. All additional files have been inspected and audited for their content as well.

The software conforms with its design goal. The software is written well, properly factored out and split into modules, and supported by test cases. No errors were found within the software itself. Errors within the testing framework of the software were addressed as part of the review process.

Any features not required or allowed by the DNSSEC Practice Statement (DPS) are only available as configuration options or not at all. Proper configuration files ensure all requirements and limitations set by the DPS are accepted.

The suggested addition of a few Assurance Claims as the claims are implemented and tested, but not claimed were added as part of the review process.

The unit tests work as expected and prove the assurances that the software can only operate within valid parameters set out by the DPS.

While not part of the functional requirements, one of the goals of releasing this software to the public is to allow the public to test and verify the software. While this is possible using one of the specific test deployment scenarios (python venv), this method is not very friendly to operators that are not intimately familiar with the software. The software in its current form is not easily installed on a regular OS such as RHEL/CentOS or Debian/Ubuntu.

Additionally, once installed, it is not easy to generate a KSK and ZSK and test the signing system, as most testing of the system depends on the existing KSK and historic KSRs. It is very difficult to confirm the system is working by generating your own keys and generating your own KSR to produce a SKR. We recommend that the software is extended a little to facilitate this use, as it will increase the number of people who will be able and willing to test the software, which will increase the public acceptance of the software.

Evaluation of Code

Structure / Organization

The software is well structured into several python modules along functional boundaries.

Linting

The setup.cfg file lists a number of ignored linter warnings. To verify no actual issues were being ignored, the linter was run with all ignore statements disabled and no issues were found.

Code Formatting

Consistency of formatting is ensured via the tools `black` and `isort`. These tools can be run via `make reformat`. Running `make reformat` reformats the imports of most python source files. In our case we needed to run this before `make test` to fix a few small layout differences. Possibly the isort ruleset has changed since the last reformatting or the ruleset is not being applied consistently across different runtime environments.

Type Annotation

Large parts of the source code are annotated with types. In some places there is added code to help the type checker `mypy` infer types when it cannot do so on its own. Running the type checker manually (`venv/bin/mypy src`) yielded 14 errors about “no type hints or library stubs”, mainly for voluptuous (only used for testing) and PyKCS11/OpenSSL. The critical code paths have been reviewed and no bugs or undesired behaviour was found.

The `make typecheck` command runs with no errors.

CI Environment

Running code formatters and linters and type checkers are properly enforced via CI systems such as `circleci` for any change proposals or contributions to the software.

Installed Linters

Tools such as `pylint`, `parsable` are configured as a linter, and properly installed as part of the dev dependencies.

Evaluation of Test Suite

The test suite utilizes the python test suite runner "green" (https://github.com/CleanCut/green) which provides zero-configuration parallel execution of the test suite. The test suite has been confirmed to run (`make test`) successfully using multi-core machine and single process (`GREEN_FLAGS=-vv -s1`).

Running Tests

Due to python pip changes in late 2020, there is an issue running tests. As the required software is versioned in setup.py, this now causes issues with 'pip' newer than October 2020. To successfully run the tests, the versioning of "**click**" was removed, and the version of "**pytest**" was set to "**< 6**".

Running `make test` discovers all 428 tests. Of these, 2 tests are skipped which require an (old) real production KSR Archive set via KSKM_KSR_ARCHIVE_PATH which was not available for testing.

Note that a change in how testing is run with pytest causes all successful tests to be marked as SKIPPED in subsequent runs. This issue is known upstream as:

> https://github.com/tholo/pytest-flake8/issues/62#issuecomment-495763052

A change is recommended in the Makefile to add "**--cache-clear**" to PYTEST_OPTS=. This ensures that in subsequent runs, previously passed tests properly show PASSED instead of SKIPPED for passing tests.

There are 112 warnings issued when the tests are run. Most of these are complaints about relative imports and DeprecationWarnings. The pytest warnings were investigated and found harmless.

Test Coverage

Script entry points properly tested

Note that these are tested in testing/softhsm/fail.sh, which is only called when running 'make fail' in the softhsm directory manually. The output contains expected errors, resulting in confusing output for anyone testing this part for proper functioning.

SKR xml file not fully validated

The SKR components are validated in memory, and then written into xml format to a file. The number of Request Bundles in the Request is verified although misconfiguration could state different numbers of received and written Request bundles in a Request. DNSKEY flags of the KSK are not verified ? DNSKEY flags of the ZSK are verified on input. There is no Assurance Claim that has ownership of the Request and Request Bundle verification of the SKR.

Assurance Claims and DPS Requirements

The software comes with a set of assurances specified in the file assurance-cases.md

The assurances are tested via integrated test cases containing claims that have been validated.

The Assurance claims should cover all the requirements from the DPS. At times, the software allows more flexibility than the DPS, in which case the software configuration should be able to produce a system that exactly matches all requirements of the DPS. The configuration file in config/ksrsigner.yaml provides a DPS compliant configuration, but has existing keys in it, so it cannot be used to test the software from a scratch configuration. It might be useful to provide a claim of Root Zone DPS compliance when the software detects a fully DPS compliant configuration file.

Missing Assurance Claims

The DPS does not place any requirements on the SKR other than the RRSIG signature validation in Section 6.8. This is a bug in the DPS. There are further "requirements" on the SKR, such as proper DNSKEY flags, same amount of Request bundles in the SKR as there are in the KSR. While there is code and test case coverage for this, there is no Assurance Claim because there was no DPS requirement listed.

Claims Validated Against Test Suite

- 1.1
  - Startup checks for all required (and valid) parameters in policy
    - 1.1.1 pass (see note above about separate fail.sh testing)

- 1.2
  - 1.2.1 verifies KSR signed by RZ maintainer
    - 1.2.1.1 pass “Test cases provide evidence that (a) a SKR with signatures which cannot be verified using the KSK is rejected, and (b) a SKR with valid signatures is accepted.”

      If previous SKR is not present, a new SKR may *still* be created if the policy configuration does not mention previous SKR location and command line args don’t specify any (after printing warnings)

      Some checks can also be disabled via policy config (warnings printed)
    - 1.2.1.2 pass
    - 1.2.1.3 validate_signatures:false policy config can disable this check (warnings printed); otherwise ok
  - 1.2.2 verifies keys+parameters are consistent according to roll over scheme
    - Note: a subsection could be added to verify the TTL, despite the fact

that this value is not used other than for RRSIG validation.
        - 1.2.2.1 pass - although key_tag and key_identifier seem to be used interchangeably
        - 1.2.2.2 pass
        - 1.2.2.3 pass calculate_key_tag could have been written closer to the notation in pre-existing software / RFC
    - 1.2.3 verifies number of keys is in compliance w/ policy
        - 1.2.3.1 pass
        - 1.2.3.2 pass
    - 1.2.4 verifies time period w/ policy
        - 1.2.4.1 pass
        - 1.2.4.2 pass
    - 1.2.5 verifies time overlaps w/ policy
        - 1.2.5.2 pass
    - 1.2.6 verifies algos and params w/ policy
        - 1.2.6.1 pass
        - 1.2.6.2 pass
        - 1.2.6.3 pass
        - 1.2.6.4 pass
        - 1.2.6.5 pass
        - 1.2.6.6 pass
    - 1.2.7 verifies domain name w/ policy
        - 1.2.7.1
    - 1.3.1 only generates SKR with valid signatures
        - 1.3.1.1 pass
        - 1.3.1.2 pass
        - 1.3.1.3 pass
        - 1.3.1.4 pass
    - 1.3.2 verifies signatures can be verified w/ public component
        - 1.3.2.1 pass (using mock of softhsm response)
        - unclaimed: it verifies the number of bundles
        - unverified: request and response number of bundles are not verified to be identical.
        - unverified: KSK parameters (algorithm 13)

Evaluation of DPS compliance

This chapter lists the related sections of the DNSSEC Practice Statement (DPS) for the Root Zone KSK Operator as published at:

https://www.iana.org/dnssec/procedures/ksk-operator/ksk-dps.html

It describes the compliance of the software to those DPS sections. The numbering in this section matches the DPS numbering. Some sub-sections are not further detailed if the section itself is out of scope for this software audit. Short technical descriptions are used to capture the requirements in the DPS. For each claim in the DPS, we list all the found assurance claims that verify these claims and note where we are missing an assurance claim.

DPS Enumeration and Assurance Claims

This is based on a properly DPS compliant **request_policy** section in the used **ksrsigner.yaml** configuration file.

        - 1 Introduction
            - No software requirements
- 2 Publication references
    - No software requirements
- 3 Operational requirements
    - No software requirements
- 4. Facility, Management and operational controls
    - No software requirements
    - Note, there are some things the software can do to facilitate some of the audit requirements, although these are technically out of scope as the human operators are responsible for the audit trail during the key ceremonies. But certain screen output of public keys, keytags, operational success/failure message could be given timestamps and software / version hashes when producing output. It does this for some operations, but not all operations.
- 5 Technical Security Controls
- 5.1.1 - key generation
    - Requirements met by FIPS certified hardware
- 5.1.2 - public key delivery from ZSK
    - in-band as zone data (RRSet list linking to known trust anchor)
        - Not supported. Or rather, indirectly supported by taking this data, and supplying it within the KSR schema in XML (see next item)
    - Via signed message
        - supported (KSR schema using verified Bundle of RequestBundle's.
        - Requirements met via Claim 1.2.* and 1.3.*
    - Human verification - out of scope
        - N/A
- 5.1.3 - RSA public key testing
    - Primality testing of RSA parameters
        - For new KSKs, FIPS certified hardware would be responsible for this

        - For new ZSKs, FIPS certified hardware is likely responsible for this.
        - For new ZSKs received via KSR, this testing is not performed by the KSK maintainer. The KSK maintainer has no signed assurance from the ZSK maintainer as part of the KSR. It is recommended that the KSK Maintainer performs this testing on all received ZSKs before allowing their use by signing the KSR. There is no Claim for this in Section 1.2.7.6 and no code in check_keys_match_zsk_policy(). This missing requirement is a bug in the DPS. It is recommended as addition to the software.
        - The ZSK DNSSEC Practise Statement can be found at: https://www.iana.org/dnssec/procedures/zsk-operator/dps-zsk-operator-v2.1.pdf . The ZSK DPS states FIPS certified hardware is used for ZSK key generation. This makes the requirement for the KSK DPS to check the received ZSK's primality not as urgent, but perhaps the KSK DPS could require the ZSK DPS provides this assurance just in case the ZSK DPS changes in the future.
    - Public exponent testing
        - Requirements met via verified evidence claim 1.2.6.4 and 1.2.6.5
        - Software has implemented our recommendation of no longer allowing exponent 3 to ensure the keys used are FIPS compliant as per https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br2.pdf
    - Key size testing
        - Requirement met via verified evidence claim 1.2.7.6 - but not explicitly in its own claim. This should be clarified explicitly in a separate claim just as was done for exponent testing.
        - Requires proper setting of **rsa_approved_key_sizes**
- 5.1.4 - KSK Key Usage Purpose
    - Only valid use of private key is
        - Signing RRset - implemented
        - Self-signing proof-of-ownership - not implemented
        - Nothing else - No other methods are implemented that use the KSK private key
    - RRsig signatures max validity period: 21 days
        - Requirement met via proper setting of **max_signature_validity**
        - No evidence claim, but mentioned as KSR-POLICY-SIG-VALIDITY, and ttest_too_long_signature_validity tests this. Probably should receive its own evince claim
    - Expiration of any RRSIG made cannot be more > 180 days in the future
        - Requirement met via verified evince claim 1.2.4.3 (proper setting of **signature_horizon_days)**
    - Emergency override should be possible to extend signatures
        - Requirement met via configuration file that can be modified for this purpose using the above two configuration options.
- 5.2 KSK Private Key Storage
    - Private keys remain only in HSM or via encrypted backup (for other HSM)

        - Requirement met by using FIPS certified hardware that only allows exporting a private key for backup using an export encryption (wrapped key) via its PKCS#11 API. Obviously does not apply when using softhsm.
- 5.2.1 KSK private key material only in HSM
    - Requirement met via configuration with FIPS certified hardware HSM
- 5.2.2 Private key (m-of-n) multi-person control
    - Enforced via HSM operation to unlock the private key for use within the HSM by the Key Ceremony.
    - The software tested has no capability of bypassing Key Ceremony
- 5.2.3 N/A
- 5.2.4 Private key backup via exported n of me scheme
    - This is part of the HSM Key Ceremony and not part of software under test
- 5.2.5 KSK Private key storage on cryptographic module
    - N/A to software under test
    - Software uses standard PKCS#11 and can use hardware or software hsm depending on configuration.
- 5.2.6 KSK private KSK key is not archived after rollover
    - N/A to software under test which never has access to private key material (it can only issue signing commands via PKCS#11 API)
- 5.2.7 KSK Private Key Transfer
    - Enforced by FIPS compliant HSM
    - Software under test has no access to private key material.
    - Any KSK Private Key backups and transfers are protected by the HSM and the Key Ceremony.
- 5.2.8 Activate KSK private key using n of m (specifically 3 of 7)
    - This requirement is met via the Key Ceremony and out of scope for software under test
- 5.2.9 Deactivate private key using n of m (specifically 3 of 7, with preference of 3 that were not used for key activation)
    - This requirement is met via the Key Ceremony and out of scope for software under test
- 5.2.10 KSK private key destruction should be permanent, secure and audited.
    - Enforced by using HSM FIPS certified hardware. Audit is done via key ceremony, cannot be influenced by software under test
- 5.3.1 KSK public keys are backed up and archived
    - software under test does not print public key when private key is destroyed.
    - This requirement is expected to be met by the Key Ceremony. Additionally, many organizations log and store all public keys used for the root zone, as this data is public.
- 5.3.2 If KSK is superseded, key will never be used to sign anything again.
    - The exact timing depends on the interpretation of "superseded". It is assumed that the Key Ceremony handles the destruction of the key via software under test. As that destroys the private key inside the HSM, the software has no way of obtaining this private key and software has no way to instruct the HSM to use a now-destructed key.
- 5.4 Activation data generation and installation

    - This requirement is met by the Key Ceremony - not applicable to the software
- 5.5 Computer Security Controls
    - N/A
- 5.6 No part of the signer system making use of the HSM is connected to any communications network.
    - It is assumed that the software under test receives the KSR as a file as per Key Ceremony (USB FLOPPY) and uses the same medium to export the SKR.
    - Software under test does ship a tool called wksr, which runs a webserver to take requests. It is assumed this method is only used for testing.
- 5.7 timestamping done manually within a 3 minute accuracy
    - Requirement met as part of the Key Ceremony.
    - It would simplify some auditing if all commands issued by software under test would display a timestamp of execution (and possible version/hash identifier)
    - Software under test cannot modify the system time
- 5.8.1 NIST SP 800-64 procedure for incorporating security and trustworthiness
    - Relevance here is the Implementation Process
    - External Code Review: This document is implementing this aspect of the security process.
    - There is documentation architectural design describing the security domains and functions maintained by the signer.
    - The architectural design demonstrates that the signer system prevents bypass of the security-enforcing functionality.
        - The configuration file allows setting "bypasses". But as long as the configuration file is considered verified with the system, then this requirement is true.
        - The system during execution does not modify the configuration file. The configuration file is loaded into an in-memory representation, which is mutable and which the test suite does modify in-flight to re-test using different options.
    - There is a functional specification completely representing the signer system and all operations associated with it.
        - The phrasing of this statement is open to debate. What is "completely representing" ? Is stating "It follows the DPS" a complete specification?
        - Aside from the issues raised in this document, we believe this requirement is reasonably met.
    - There is a modular design description and a one-to-one correspondence with the modular decomposition of the implementation.
        - The phrasing of this statement is open to debate.
        - Aside from the issues raised in this document, we believe this requirement is reasonably met.
    - The implementation representation completely and accurately implements the security-enforcing functions.
        - Aside from the issues raised in this document, we believe this requirement is met.
    - There is a process to confirm no software has been tampered with

        - N/A for software under test
        - Presumed to be done as part of the OS / software install and before each use of the OS / software via the Key Ceremonies.
- 5.8.2 Software management controls
    - Process to control and monitor configuration of the system
        - Out of scope for software under test, although it could present hashes of any configuration file it opens. It might be redundant but possibly provide an additional verification there was no "off camera manipulation" of configuration files.
    - creation of hashes of all software and software updates
        - Out of scope for this software
    - software is manually verified before its use
        - Out of scope for this software
- 5.8.3. Life cycle security controls
    - Minimum of maintenance desired - so no critical updates would be needed
        - Requirements met within reason. There could be updates to the DNS protocols and algorithms and the DPS, all of which would require updates of the software. For example, if the root zone decided to switch to the ED448 algorithm, or if it would switch from NSEC to NSEC3 or if it would set a new DNSKEY flag (such as the proposed DELEGATION_ONLY proposed flag)
        - Algorithm rollover support would be nice, but mostly for the software to work with non-root zones. The same is true for CDS / CDNSKEY support.
        - New DNSKEY flags are not supported, but arguably those cannot be supported without the risk of an undefined DNSKEY flag being accepted as valid.
        - An increased RSA key size is not supported. It would be unlikely to be required, but it could be imaginable that some new attack would require increasing the RSA key size from 2048 to 4096, It seems reasonable this would not be expected to be within the minimum maintenance requirements of the software.
    - Origin of all software and firmware will be securely authenticated
        - The origin check might not be enough, as for example a Debian signing key could provide origin verification of packaged software, but does not guarantee against a compromise at the Debian Project servers or their software signing key. It also does not protect against compromise of their packages. However, targeting an offline signer that uses this software through such "upstream" compromise would be extremely difficult.
        - The software requires many python modules. Especially if this software is released to the public, these requirements become public knowledge and turn into attractive targets for malicious modification. Malicious modifications in these required python modules are easier to accomplish. For instance, a debian or PyPi package maintainer for one of the python packages could be bribed to include a backdoor into a newer version of that software. The Debian project signing key

would still sign this new (now malicious) version of the official upstream package. These modules should be audited for malicious use specifically targeting an offline signer. That would limit damage to only breaking DNSKEY RRsets or their signatures, introducing a rogue DNSKEY, or rogue modification of signing configuration files. That severely limits the audit required on those modules.
        - This makes it vitally important that the signer once built, is never connected to the internet or WAN for its entire life cycle.
        - Any DoS attack would not accomplish much, as the keys are generated far in advance of their deployment, giving all parties involved in signing the Root Zone ample time to deal with such malicious software modifications.
    - Hardware procurement requirements: N/A
- 6. Zone Signing
    - Note the DPS contains confusing language in this section
    - Provide a signed DNSKEY RRset for the root zone with the ZSK's provided and the appropriate KSK's. This is the high level requirement of the software
    - Daily root zone signing: N/A
- 6.1 KSK must be 2048 bit RSA key
    - Requirements met by software via configuration file.
- 6.2 NSEC must be used by the RZ ZSK Maintainer
    - While the creation of NSEC records is done by RZ ZSK Maintainer, the RZ KSK maintainer needs to provide the proper DNSKEY algorithm that supports NSEC. For algorithm 8 and newer, DNSKEY algorithms support both NSEC and NSEC3. The Root Zone is currently using algorithm 8. The RZ ZSK Maintainer could switch from NSEC to NSEC3 without permission of the RZ KSK Maintainer, but this cannot be prevented by the software. Similarly, opt-in could be enabled for the Root Zone if a switch to NSEC3 is made. By stating that NSEC is required, this means opt-in is also not allowed to be used by the RZ ZSK Maintainer.
- 6.3 RRSIG signature format
    - Requirement met via configuration file options for dnskey algorithms.
- 6.4 ZSK rollover quarterly
    - Performed by the RT ZSK Maintainer and RT KSK Maintainer
    - RT KSK Maintainer must sign appropriate DNSKEY RRsets. This is enforced via configuration file by the software under test by various timing constraint values.
- 6.5 KSK rollover
    - KSK rollover within 5 years
    - Note: This requirement has not always been followed in the past, so this requirement cannot be strictly enforced by software under test. Since extending this period is not immediately harmful, and could be useful in case there are valid reasons to continue operating beyond a 5 year span (eg during a pandemic), it makes little sense to enforce this upper limit in software. The software could enforce this via configuration file, which could be modified in case of emergencies, but the value of this software change would be very low.

    - The software could refuse rollovers that are technically incorrect and/or unsafe. For example, by rolling faster than the TTL of the DNSKEY RRset. While this is enforced for ZSK rollovers, which timings are under strict controls, this is not the case for KSK rollovers. However, this is enforced via Key Ceremonies as DNSKEY RRsets are not created for immediate deployment, and human auditing would (hopefully) catch these.
    - KSK once retired should never be used again for anything. Destruction will only happen at the next key ceremony, so it will be left unused (but the entire OS / system will be unused too). Technically, this might be a violation of the DPS, and should probably be updated in the next version of the DPS.
- 6.6 KSK lifetimes
    - From the 1st of Jan, Apr, Jul and Oct for approx. 90 days
    - 90 days cycle divided in 10 slots of 9 days
    - last slot in the cycle fills up any unallocated days
    - For each time slot, sign RRset with 21 days (overlapping 11 days)
    - In case of KSK roll,
        - Slot 1 normal operation, current KSK, current ZSK and old ZSK
        - Slot 2-6 introduce new KSK, signed with current KSK, only current ZSK
        - Slot 7, sign with new KSK, still includes old KSK, current ZSK
        - Slot 8, sign with new KSK, set revoked bit for old KSK, current ZSK
        - Slot 9, sign with new KSK, old KSK removed. current ZSK and new ZSK
    - Note: in the previously used KSK roll, there was a deviation from this process due to receiving an unexplained larger amount of traffic than expected. The DPS (6th edition) was updated to formally allow this as a result of this finding.
    - Requirements met via verification of slots (RequestBundle) inside the KSK Siginng Request (KSR) by the software.
- 6.7 ZSK verification
    - Each Key Signing Request (KSR) is signed with current ZSK as proof of ownership of private key
        - The exact phrasing of the DPS in this section is unfortunate. It states the KSR should be signed by the ZSK. But the KSR itself is not signed and verified by the software. It is verified out-of-band by humans and an OpenPGP signature (presumably before and during the Key Ceremony). The slot 1 KSK RRset is signed by the current ZSK, and this does count as a proof of ownership of the private key. Furthermore, further DNSKEY RRsets are signed by the ZSK in a chain, moving the trust in the future/current ZSK one step at a time.
        - The oddity of this system is that the ZSK signs a DNSKEY RRset that is actually never meant to appear in the RZ (it does not contain the KSK DNSKEYs). Whether that is more or less safe to accidental or malicious use than the ZSK being used to sign non-DNS data (eg the KSR) is debatable.
    - Software validates signature and algorithm parameters of all keys in the signed RRset.
        - supported (KSR schema using verified Bundle of RequestBundle's.

        - Requirements met via Claim 1.2.* and 1.3.*
    - Secure transport of the KSR is done out-of-band by humans during the Key Ceremonies and not by the software under test.
        - The software implements kskm-wksr as a secure transport mechanism. It is assumed this method is only meant for testing and not production use. We would not recommend using a non-hardened python based TLS web server. Regardless, this tool cannot certify the transport from within the organization to the secure offline signing server. The Key Ceremony performs the validity checks there.
    - The software tests the validity of all the KSR elements against its local policy in the configuration file and performs Proof of Private key of the ZSK Maintainer by checking the RRSIGs (<SignatureData> element) made by the ZSK public keys inside the KSR.
    - The software signs new DNSKEY RRsets and delivers these in a Signed Key Response (SKR)
- 6.8 Verification of signed RRsets
    - Produced signed RRsets must be DNSSEC validated
        - Requirements met via Claim 1.3.2.1 (SKR-VERIFY)
    - Integrity of signatures in SKR verified by RZ ZSK maintainer
        - Out of scope for software under test
    - The DNSKEY RRset could validate properly, but if all ZSK's have the REVOKE bit set, the resulting Root Zone data itself would fail to validate.
- 6.9 TTLs used by signer system
    - DNSKEY RRset: 48 hours
        - Requirements met via dns_ttl or ksk_policy.ttl configuration value.
        - It would be an enhancement if the written SKR xml would be read and parsed and items like TTL and DNSKEY flags would be explicitly covered by an Assurance Claim
        - Note that the TTL that is received in the KSR is ignored other than for converting the data to wire format to verify the signatures. If the received TTL conveys the ZSK Maintainer's desired policy, and it would deviate from the KSK Maintainer policy, the KSR should be rejected or at least present a warning that the value will not be used for the SKR.
    - NSEC RRs - same as SOA minimum (24 hours)
        - Not applicable for software on KSK signer - only for ZSK signer
    - DS - 24 hours
        - Not applicable for software on KSK signer - only for ZSK signer
    - RRSIG - same as covered RRset
        - Only applicable for DNSKEY RRSIGs for the KSK signer
        - Requirements met via configuration of the ksk_policy.ttl
- 7 Annual audit of RZ KSK systems
    - Out of scope for this deliverable
- 8 Legal matters
    - Out of scope for this deliverable
- References
    - N / A

Clashing Key Tags

The DPS does not describe that it wishes to avoid a Key Tag clash between KSK and ZSK. As such, no Assurance Claim is rejecting any received ZSK's that have the same Key Tag value as the KSK.

It is recommended that an Assurance Claim and supporting code is added to prevent clashing Key Tags, despite that this is currently not a DPS requirement.

Testing of the Script Tools

The tool binaries (python scripts) are not tested themselves, only the libraries they call are tested. In practice, there is minimal glue between the binaries and the library functions it calls. Nevertheless, we think it would be useful to cover the script tools.

Testing of kskm-keymanager

This tool has no formal testing itself, but it is called in testing/softhsm to create a mock hsm for the test suite of the python library functions.

The tool, once installed, is easy to use.

Testing of kskm-keysigner

The tool, once installed, is hard to use. Especially when attempting to start from "scratch".

Testing of kskm-sha2wordlist

The tool is not used by the software tested. The library code is uses it used by other library functions. The tool could be used during a Key Ceremony to verify specific files, as it produced easier pronounceable output that similar tools like sha256sum do not.

Testing of kskm-trustanchor

This tool has no formal testing itself, but it is called in testing/softhsm to create a mock hsm for the test suite of the python library functions. The tool is not used by the software tested. The library code it uses is used by other library functions. The library code this tool uses is only partially tested and has no Assurance Claim associated with the library code. The tool could be used during a Key Ceremony.

Testing of kskm-wksr

The tool and its library code are not part of any Assurance Claim or testing. It is basically not part of the software under test but used to make testing easier. It might also be of use for third parties that want to use this software but in an online fashion instead of a full offline fashion. Only basic testing was done, such as confirming the webserver requiring a client side TLS certificate uses proper certificate validation.

Suitability for Third Party Users

Functional for Root Zone Only

We could not create SKR using any domain signer name other than “.”. Attempts fail with several NotImplementedErrors.

```
NotImplementedError("Non-root dn2wire not implemented")
NotImplementedError("Non-root dndepth not implemented")
```

First Run Experience

The barrier of entry / “first run experience” of this software is not optimal. A potential user needs to provide a suitable mac or linux environment, modify it by installing the correct python version, swig, and softhsm.

After providing these, the test suite can be run. Running the `kskm-*` commands still require providing configuration files that need to be copied from templates and modified.

A “low effort” installation experience could be provided using Docker or Vagrant. The potential user could install this software, running in a default configuration using preconfigured softhsm. It would exist within a virtualized/containerized environment on the developer’s desktop, with little risk of breaking or bloating their regular environment.

Software Installation

The software was provided via a private git repository. It is fairly specifically tailored to a specific containerized version of an Operating System called COEN (https://github.com/iana-org/coen).

This presumably has been done for the purpose of creating a reproducible OS for the hardware signer and to create a specific virtual environment for unit testing. During our testing we found a number of issues that made it harder for us to test the software outside of these deployment scenarios:

- Makefile should not hardcode specific python version (python3.7)
- specific versioning of requirements makes it incompatible on any other operating system or even with up to date versions of “pip” when generating the **venv** environment.

With these issues addressed, we managed to create a functional rpm package for a CentOS based system as proof of concept that the code runs generally on any Linux system.

An rpm based packaging effort was done to better understand the software dependencies and usage. The tools however, cannot operate on default directories for configuration (eg.

/etc/kskm) or operate on directories for their data (eg /var/lib/kskm). This severely constrains their usability with testing or actual deployments for other zones.

Installation

There is no real formal installation target or software installation method.

Documentation

It would be useful to have documentation on how to take an existing KSK private key, and import it into the system. Perhaps add a kskm-keymanager import command that takes a zone name and encrypted key file, and generates (or displays) a proper ksrsigner.yaml

Usability of the Command Line Tools

To encourage people testing the dnssec-keytools, it needs to be very easy to use. It should be easy to package this software, install it and run it without further virtualization or containerization. For example, think about a dedicated raspberry pi Linux machine acting as (offline or online) kskm machine.

Comments Applying to All the Command Line Tools

Usability of the command line tools in generic deployments - eg regular people using these tools for evaluation or even actual production machines that are not virtualized - is not easy.

The command line tools have no man pages

The tools only have the current work dir as default work directory. This does not work well when packaging the tools on a system where a dedicated user/homedir would be used (eg /var/lib/dnssec-tools)

The help messages show:

```
--hsm HSM      HSM to operate on (default: None)
```

What does "None" mean in the context of softhsm? Can it work without softhsm? Is softhsm not an hsm? Perhaps default to softhsm ? What value is HSM? Is there a list of names of valid values? Is it a config file? A PKCS#11 library file ? It seems None picks up the softhsm configuration.

kskm-keymanager

kskm-keymanager only looks for its yaml file in the current working directory. Perhaps support for ~/ or /etc/kskm/ could be useful (eg "compile" time variable)

kskm-keymanager creates a logfile per invocation, causing a littering in the current directory the tool is used in. There should be a method to configure (and default) to a logging directory (eg /var/log/kskm/)

Generated logfile names are unpredictable enough for humans, but too predictable for computerized race conditions, although if going by the assumption that no malicious code can be loaded onto these machines, that should not be an issue. If the tools see wider use on online signers, this might become an issue.

The tool does not (easily) support starting from scratch with no keys. It requires importing the new key into softhsm, then populating it with key: sections with bogus keytag/dsrecord, disable signature verification, then fixup the entries based on the kskm-keymanager errors.

Some errors still throw hard to read python backtraces, such as running the following command twice:

```
kskm-keymaster keygen --label test_rsa --algorithm RSASHA256 --size 2048
```

It is not obvious where the zone name can be configured (e.g. instead of using "."). We found that in some places in the code, the root zone is assumed, whereas in other places in the code, zone names seem to have been intended to be supported.

Example use of manual test after creating proper softhsm.com and ksrsigner.yaml files:

```
kskm-trustanchor
kskm-keymaster inventory
kskm-keymaster keygen --label test_rsa --algorithm RSASHA256 --size 2048
kskm-keymaster keydelete --force --label test_rsa
```

Perhaps a tool could be written to kickstart the process along with the softhsm initialization commands (kskm-init ?)

kskm-ksrsigner

Throws a python exception instead of a human readable error, when config file not found

Minor Complaints

Immutable python objects

Several validations rely on immutability/hashabilty of classes from kskm.common.data (Key class, for example, used to check if a key is in a bundle like `key in some_list`). The hashability is currently ensured by `@dataclass(frozen=True)` but no test case verifies that this is how it is done to prevent a less safe hashing from being implemented or that some fields are excluded by hash=None.

Format of the KSR and SKR Files

It is understood that the KSR and SKR formats are currently in production use, and not planned to be updated for enrollment of this software. As thus that the software cannot be modified to use an improved format.

The naming of Request and Requestbundle is counter-intuitive, the Request is the bundle, and the bundle is a single sign request with elements that are now considered a bundle.

Why is the TTL set per key and not set in the RequestPolicy? The TTL of each DNSKEY should really be identical from a DNS point of view. From a DPS point of view, these TTLs MUST be identical as the TTL of the RRSIG must match the TTL of the DNSKEYs, which can only be true if the TTLs of all DNSKEY's are identical.

The ZSK signs valid formed DNS data that is never meant to be published in the Root Zone as a method of Proof of Private key. Perhaps it is safer that it signs something that could never accidentally be included in the Root Zone as properly formatted DNS data. The counter argument is that signing something else might possibly lead to other attacks.

The splitting up of DNSKEY RR into its XML components is awkward. It needs to be converted to a DNS RRset in presentation format, to then be converted to wire format, to then be checked by DNSSEC validation software, seems less than ideal. It helps neither automated processes nor humans.

The SKR transports the RRSIG values covering all DNSKEY RRs, but the XML only includes the KSK DNSKEY elements and not the ZSK DNSKEY elements. The SKR itself cannot be used to recreate a valid DNSKEY RRset with RRSIG record in DNS presentation format.

Further verification and extraction software is required. The formats would have been much simpler if DNS presentation format was used throughout the system to convey DNSKEYs and RRSIG records.

Required Libraries for the dnssec-keytools Software

An audit of the Operating System (OS) and required python libraries is out of scope for this audit. However, an inventory and brief evaluation of the python libraries used has been included.

The software is written in python, and requires a number of libraries. While some were specified in the documentation, others were only listed inside code (setup.py).

It seems that the list of required versions of these additional python libraries is based on what was available for the intended signer hardware operating system (COEN).

These requirements make it harder for independent third parties to test the software manually, outside the specifically tailored python venv environment. And from October 2020 onwards, it will fail due to changes in the pip software and manual editing of some of the versions is require. It is understood that publication of the software is meant to encourage people to test it, in which case making it easier to install on different platforms would likely increase the number of people testing out the software, or even deploying the software for their own zones, converting their online KSK+ZSK signers, into a system with an offline KSK signer.

While a cursory audit of these general purpose libraries might be prudent, it is unlikely that any vulnerability or malicious code embedded in these libraries could cause the signer to create bogus or malicious output. Since the signer is not connected to the internet, and its only output is the SKR containing signed RRsets, any malicious library would have to specifically target the signer python code. While this signer code will be published, adding attack code to these libraries that are used for many other projects without being detected either upstream at the maintainer or downstream at the OS vendor would be unlikely. These unlikely events are further prevented by strictly following the key ceremonies and the DPS requirements, which will ensure a number of very knowledgeable humans will be looking at the signed DNSKEY RRsets produced by the signer.

More invisible attacks, such as causing the signing machine's randomness to be weak, have no effect on the signer, as all private keys are generated by HSM hardware and are never exposed to the signer machine. The key ceremony again protects against bypassing the HSM if malicious code would cause weak or known private keys to be generated on the signer in software instead of via the FIPS. As the signer code uses a PKCS#11 library that only talks to the FIPS hardware, an exploit would basically have to replace a large chunk of code.

For this reason, we do recommend that the softhsm library used for testing is not installed on the production signer machine, so it cannot be accidentally or maliciously be used instead of the FIPS hardware.

As the signer machine is completely offline, the only potentially dangerous input comes from the Signed Key Request (SKR). While the signer validates this request, it would be prudent to verify its contents before placing this file on the signer hardware.

Suggested Changes

Below are listed the suggested changes that No Hats has identified:

Required

- None

Recommended

- Support a system-wide or home directory based configuration file
- Support system wide or home directory based data storage directory
- Support a system wide or home directory based logging directory
- Prevent clashing Key Tags between KSK and ZSK
- Place all configurable (and even hardcoded) values into configuration files
- eg don't place them in config_misc.py, such as rsa_approved_exponents
- Provide a clear configuration file that implements the DPS for the Root Zone
- Improve the software to make it easier to install on regular Linux systems to not discourage people from testing or even deploying for their own zones.
- If the DPS is updated, please find other acronyms or terms for KSR and SKR.

Nice to Have

- Ensure the domain name can be configuration in the configuration file, and that it will be properly used through the python library, so the software can be used for non-root zones
- Provide a clear example of how to run things, including creating a KSK from scratch, without prior trust of any ZSK, using files as transport medium (not webserver)
- Provide a tool that can read KSR and SKR and convert these to DNS presentation format using DNSKEY and RRSIG records.
- A complete set of test vectors that include all private keys for KSK and ZSK to simply running tests by making changes to an KSR without needing to disable signature verification at various places.

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

The software version ("Software") that No Hats has reviewed can be
identified by the following repository values:

ssh://github.com/iana-internal/dnssec-keytools-2019-dev.git

commit efd2237dd48e58cd0a621d0b0382b5217002a3ac
Merge: 07cad20 8c9ee14
Author: James Mitchell <james.mitchell@iana.org>
Date: Fri Apr 9 11:19:32 2021 -0700

    Merge pull request #28 from iana-internal/doc-update

No Hats and all of its personnel that performed work on this contract
have not been, directly or indirectly, involved in the creation of the
Software. No Hats has reviewed all the source code, documentation and
other files that came with the Software. The complete list of engineers
who worked on this audit are:

Paul Wouters   paul@nohats.ca
Martin Ringehahn   martin@nohats.ca

The audit was performed against the 5th edition:

https://www.iana.org/dnssec/procedures/ksk-operator/ksk-dps-20200407.txt

As a result of this audit, the DNSSEC Practise Statement was updated
to the 6th edition:

https://www.iana.org/dnssec/procedures/ksk-operator/ksk-dps-20201104.html
-----BEGIN PGP SIGNATURE-----

iQIzBAEBCgAdFiEE84Tnih0zUtPrkoq4YtNYL+D9lNIFAmB3iXMACgkQYtNYL+D9
lNKZNw/9HtZO3oetBu52eqx3/v3hv5rSHdS6hRyk862gWyPO5U1g4AO/YjltD/IE
rCmMzp8/IXXAxLghVTVFeagqwr4O8lFCXtNFVOwDV5aEbXfyqXhYknfKQJhnVCY5
TRAyRqEBcspiNrdPaD6B1Hw6WkAUow2WxZYP72oNlBmrNSaPuLlmnrU9V4HL99Fk
hRfCVYUJ6Z1+JI1QPOYk64ddO00mXL+oTzK05d9ZfavdwbJTgHVav5HQiLTYQOlQ
JewhBhjQyjkX0Wb7kB1LHLW9P2T3pxzFQjXTBdvuliE++CM6T9TwSkZd6Zr4EuUc
4OtWtzJTDCSXD80VCDjs5KhaPpX0FDzv94y9UwF4u1toBIXvCoIJ0VPQKqTzZr4Y
/Z1pw+GyMa01geVfvI9HCJg1Lk/twWnrmnZg6fowtoPz/6b5ninp6v0Vo8ezHWpu
u3BRG0P9IQflnUW/JIzfTYXuXYUlcz9UHbS9UHyMHAMTTG4UwjB3O6lGh+9TFrY9
B6AALdiIpYCBxSy20FhhhjfmGPrt4vBA5OMJWCLwRw3kqzlBW1d4nazgk3S7Oq7f
seQrVVWnpBm2vP/mz5B4mYpVennEpk+BDEThv10yx9er/aR5CEs2kufY+gVeWwhL
oxOoY4c66zEJx0rRgEEgeIXPXWhhg1ntChGJ5FG2mSZNXlqwG1k=
=Fc4B
-----END PGP SIGNATURE-----