Appendix B

ToDo:

Introduction

Secure generation and handling of private key material is critical to secure operations of a federation. The compromise of the key could lead to untrusted metadata appearing to be trusted and vetted due to a forged signature. Key generation and handling must be done in a way that the key is secure, but also recoverable for disaster recovery and business continuity.

Entropy source

In order to generate an RSA key pair, a sufficient source of random data is needed. NIST guidelines[1] help with this. There are many potential sources for random data, including software, hardware, and external sources. Most operating systems provide a source which can be used (e.g. /dev/random). Some hardware platforms provide a source implemented in silicon or other means (e.g. Intel RDRAND[2]). There are also inexpensive commercial hardware devices for generating source data[3]. Natural phenomena can also be used as a source for random data, and there are some services available that provide this data such as NIST Randomness Beacon[4], random.org and hotbits[5]. An additional source of random data is the AWS CloudHSM[6]. Public sources of random data SHOULD NOT be directly used to generate actual key material. E.g. the key should not be ***directly*** derived from public random data, since it is by definition public. It can be used for generating entropy, however.

Test suites are available that will analyze the quality of the random data source, such as the NIST's Statistical Test Suite[7]. In Linux, there is rngtest[8] which will test the PRNG output against NIST FIPS 140-2. An additional open source test suite (dieharder) is available for Linux[9]. At key generation time, rngtest is a practical method

---

[1] https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-133.pdf
[2] https://en.wikipedia.org/wiki/RdRand
[3] https://www.amazon.com/TrueRNG-V3-Hardware-Random-Generator/dp/B01KR2JHTA
[4] https://beacon.nist.gov/home
[5] https://www.fourmilab.ch/hotbits/
[6] https://docs.aws.amazon.com/cloudhsm/latest/userguide/openssl-library.html
[7] https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software
[8] https://manpages.debian.org/jessie/rng-tools/rngtest.1.en.html
[9] http://www.phy.duke.edu/~rgb/General/dieharder.php

of testing the quality of random data as it is a readily available utility in Linux. Other test suites could be used to test the quality of random data produced by the hardware/OS before or after key generation.

We assume in this plan that a Cryptographically Secure Pseudorandom Number Generator (CSPRNG) will be used for the key generation, and that it will be tested using one of the test methods for ensuring suitable “randomness” before it is actually used to generate the key.

While running, an operating system attempts to gather “entropy” from various sources, which is used to seed the CSPRNG. The longer a system is running, and the more activity on a system, the more entropy it is generally able to gather. In Linux, the available entropy pool can be checked by examining the file /proc/sys/kernel/random/entropy_avail. There are blocking and non-blocking PRNGs, and if a blocking PRNG is used when this source is exhausted, the PRNG will block until there is a suitable supply of entropy to continue. This blocking vs. non-blocking is a subject of much debate[10]. For purposes of this document, we’ll assume that the blocking source will be used, and a key will not be generated unless there is a sufficient pool of entropy.

Various means can be used to regenerate the available entropy pool once exhausted over and above the normal means of regular system activity. This can be useful on systems that are generally idle or in such cases as a VM that does not generate much entropy on its own. While public random data sources should not be used to directly generate key material, they may be used as sources of data to generate activity on a system, which would then increase the available entropy pool on the system. E.g. one can perform functions on a large store of random data (md5, strings, dd, etc.) to generate activity on the system, thus generating available entropy.

If available entropy becomes an issue during key generation, we may want a pre-stored source of random data for these purposes. Testing so far demonstrates this may not be needed.

Key generation

OpenSSL is often the de facto tool for generating RSA public and private keys. OpenSSL has had some security issues in the past, but is under constant improvement due to its ubiquitous use in many cryptographic implementations. There are alternative tools to OpenSSL for key generation, however it is beyond the resources of this group to evaluate the cryptographic strengths or weaknesses of all of the alternatives. Therefore it is the position of this document that OpenSSL, being the de facto standard tool, will be used for key generation.

The possibility of generating a root key and intermediate signing key was considered for metadata signing as part of this procedure, but was discounted for several reasons noted in Appendix A to this document.

[10] https://www.2uo.de/myths-about-urandom

Key generation location

One concern for key generation is the location in which it is performed. There have been demonstrated attacks via acoustics, radio frequency (RF), power monitoring, etc. for side channel attacks against key material. While it can be assumed this would be an extremely rare attack, we should plan to account for such attacks in order to prevent them, however unlikely it may be.

One method for preventing such an attack would be to generate the key in a secure facility or Faraday cage. Another method would be to generate the key in a remote location, far from any eavesdroppers or cellular/wifi signals. The remote location should not be pre-arranged and chosen only shortly before key generation. The location should also be such that a threat to the security of key generation would be easily observed.

Key handling after generation

Once a key is generated, it must be securely handled. Ideally, the key would never be able to be exported or viewed in an unencrypted form. There are drawbacks to this, such that the key is never portable from the device that generated the key. This results in vendor/hardware lock-in. In order to handle the key in a secure manner, no one individual should be able to recover the unencrypted key material. Tamper proof/resistant storage should be used for all long term storage of material. Chains of custody should be documented and maintained.

Transport of key into HSM

This is one of the most critical phases of getting to a production signing key in a CloudHSM. Because the key has to make it into the HSM in a usable format, and it must be transmitted across the network into the cloud, risk for compromise is high during this process. Several measures can be taken to mitigate this risk, however. End-to-end encryption of traffic between the AWS client utilities protects the traffic between the app and the HSM. SSH tunneling will additionally protect the transport layer. Potential use of a non-user laptop that gets wiped before/after the process ensures against eavesdropping of attackers due to a compromised OS. Use of a dedicated SSH key and freshly created VM in AWS ensures against prior compromise of credentials and VM in AWS.

Protection of key on HSM

A key imported to an HSM, such as our signing key, will be exportable. If the key were generated on the HSM, it could be marked as non-exportable, however that means the key is forever locked to that HSM solution. Therefore, some other means of protecting the key from malicious export must be used. We will do this through a combination of a specific user account for key ownership, key sharing to a separate key user account for cryptographic functions, and quorum authentication for manipulation of the key-owner account.

This is done through key-owner and key-user Crypto User accounts, and individual HSM Crypto Officer accounts (e.g. xxxx, xxxx, xxxx, etc.). These account types each have different permissions enforced by the HSM[11], as seen in the table below. The key-owner account will have ownership of the key which implies export permissions, but it will have its password randomized and not recorded after key importation and sharing, thus restricting access to this functionality. The key-user account will have a known password, but will not be able to export the key. This inability to export the key is enforced by the HSM. The key user will only be able to use the key to perform singing, encryption, etc. The HSM will require a quorum of users to authenticate in order to change the randomized password on the key-owner's account if the key ever needs to be exported, or if additional keys need to be generated or imported at a future date. *Key exportation should be impossible without quorum.* In the event that enough Crypto Officers are no longer present and in a position to be able to perform needed operations, for the purposes of disaster recovery, a new key owner, key user and crypto officers could be created on a new CloudHSM, and the existing key could be imported. A table of user permissions from AWS CloudHSM documentation is pasted below.

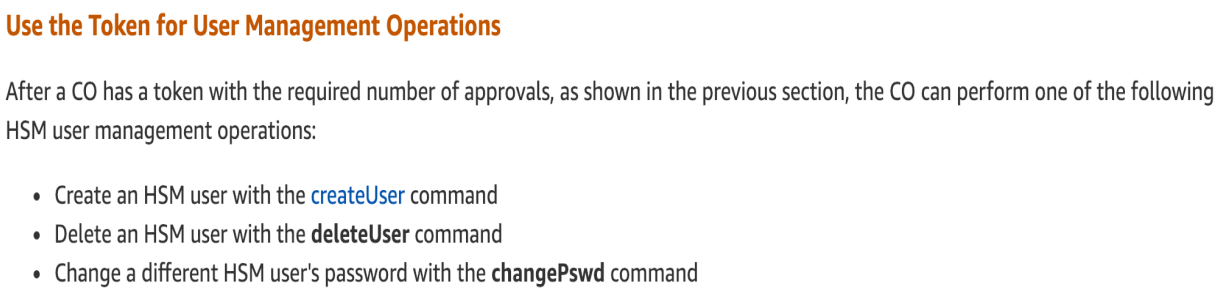
HSM User Permissions Table

The following table lists HSM operations and whether each type of HSM user can perform them.

	Crypto Officer (CO)	Crypto User (CU)	Appliance User (AU)	Unauthenticated User
Get basic cluster info[1]	Yes	Yes	Yes	Yes
Zeroize an HSM[2]	Yes	Yes	Yes	Yes
Change own password	Yes	Yes	Yes	Not applicable
Change any user's password	Yes	No	No	No
Add, remove users	Yes	No	No	No
Get sync status[3]	Yes	Yes	Yes	No
Extract, insert masked objects[4]	Yes	Yes	Yes	No
Key management functions[5]	No	Yes	No	No
Encrypt, decrypt	No	Yes	No	No
Sign, verify	No	Yes	No	No
Generate digests and HMACs	No	Yes	No	No

Once quorum authentication is enabled, all user modifications (create, delete, password change) require a configurable minimum quorum in order to be performed. Additionally, modifying the minimum required quorum requires a quorum.

[11] https://docs.aws.amazon.com/cloudhsm/latest/userguide/hsm-users.html#user-permissions-table



Use the Token for User Management Operations

After a CO has a token with the required number of approvals, as shown in the previous section, the CO can perform one of the following HSM user management operations:

- Create an HSM user with the createUser command
- Delete an HSM user with the **deleteUser** command
- Change a different HSM user's password with the **changePswd** command

The individual quorum RSA private keys used for quorum operations are stored by each quorum member. It is imperative not only that these keys be stored securely, but also that they not be lost. We will need to establish practices around the individual storage of these quorum private keys, and we will need to account for possible employee changes over time.

Use of key in signing operations

The HSM will perform singing operations using the protected private key based on successful authentication of the key user account to the HSM. The HSM can only be communicated with via a Virtual Private Cloud (VPC) network within AWS. There are various means of communicating with it from external locations (e.g. laptop -> SSH tunnel -> VM -> VPC -> HSM), however it is not directly reachable from external sites.

Authentication to the HSM is username/password based for users (Crypto Officers) and also for the key user. This username password pair must be protected. For automated signing operations, the credentials for the key-user will need to be stored by the device/instance requesting the signing, however, care must be taken in doing so. Compromise of the container or host could allow an attacker to fraudulently sign material, such as metadata.

Compromise of the key user credentials will not allow for compromise of the key itself. If the account credentials are compromised, the attacker would also have to use specific software/methods to be able to submit fraudulent signing requests in addition to knowing the credentials, and would have to have some form of access to the VPC. It is not a likely scenario, though the possibility must be accounted for and compensating controls must be in place. The HSM also generates strong audit logs by default. These logs will need to be exported from AWS into an additional log infrastructure.

Long term storage of key backup for DR

The key itself will be encrypted via GPG/AES with a randomly generated symmetric passphrase approaching 256 bits of entropy. It is highly unlikely that the key could be decrypted simply by accessing a copy of the stored and encrypted key. Current estimates are that it would take 3x10^51 years to brute force an AES 256 bit

key[12]. Therefore, we should be able to store multiple copies of the key in physically protected, distinct geographic locations. If needed, the encrypted form of the key, by itself, could be duplicated or transported to a new location without serious concern.

The key decryption key is then sharded using a quorum mechanism; Shamir's Secret Sharing Scheme[13]. It is beyond the scope of this discussion to delve into the details of the cryptography used by SSSS, however it seems to be an accepted method of sharding a key into a quorum scheme.

CD-R and DVD-R media SHOULD NOT be used for long term storage due to degradation of the dye used in creation of data tracks on the media (both) and susceptibility to scratching (CD-R). Solid state (SS) media (USB flash drive or SD card) is more acceptable. A printed ASCII armored copy MAY be stored with the electronically stored copy in the same tamper proof enclosure. E.g., in each storage location, and in the same tamper evident envelope, the key could be stored on a thumb drive with one printed copy.

2048 vs. 3072 vs. 4096

We assume that we will move to a 3072 bit key for production metadata signing as the generally accepted key size for federations. We also generate a 4096 bit key for future use if needed. We also assume that we will not generate a 2048 bit key. Elliptic Curve keys are problematic at present, and it has been decided that EC keys are not necessary to be generated at this time. Future possible generation of an EC key will require research since it is not as straightforward as generating an RSA key with the OpenSSL tools.

Key generation procedure

Equipment used

1) Hardware for key generation
    a) Operating system on clean removable media
    b) Ideally no permanent hard drive installed
    c) Wireless interface (including bluetooth) disabled in boot configuration
2) OS for key generation
    a) Hardware specific Debian derivative
3) Solid state storage devices
    a) SS flash memory for encrypted keys
    b) Separate SS flash memory for shards
    c) Number of copies:

---

[12] "Breaking a symmetric 256-bit key by brute force requires 2^128 times more computational power than a 128-bit key. Fifty supercomputers that could check a billion billion (10^18) AES keys per second (if such a device could ever be made) would, in theory, require about 3×10^51 years to exhaust the 256-bit key space."
https://en.wikipedia.org/wiki/Brute-force_attack

[13] https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing

i) Minimum of 2 for encrypted key output
ii) Minimum of 2 for each shard
iii) Media for short term encrypted keys and short term key (4 USB sticks)

4) Video recorder and tripod
5) Portable fireproof or similar safe
6) Tamper-evident envelopes, minimum of 14 (8 for SSSS shards, 2 for long term storage encrypted keys, 4 for short term encrypted key and individual short term shards)

Process

Initial key generation

Set up video recorder to record entire process start to finish. Pre-print copies of the procedure, with space for each individual present to initial that the step was completed as written. Include signatures/dates/printed names of individuals at the end of the document.
Key generation laptop:

1) Before booting hardware
    a) verify checksum on image
    b) write image to boot media
    c) modify configuration to disable wireless devices
    d) Have available thumb drive or MicroSD with secure-delete, SSSS, and rng-tools packages available
    e) On above thumb drive, have blocks of available random data for entropy generation
2) Boot hardware from media
3) Inspect /proc/sys/kernel/random/entropy_avail
4) Install rng-tools, ssss
    a) Verify checksums of packages (pre-printed)
    b) Run $ rngtest -c 1000 < /dev/random
    c) Record results
5) Inspect /proc/sys/kernel/random/entropy_avail. If entropy is not suitable, generate entropy
    a) md5sum the install media
    b) md5sum block of random data obtained previously
    c) Once entropy is available, proceed
6) Generate private/public keypair(s)
    a) openssl req -x509 -sha256 -nodes -days 7302 -newkey rsa:3072 -keyout /tmp/privateKey3072.key -out /tmp/certificate3072.crt
    b) openssl req -x509 -sha256 -nodes -days 7302 -newkey rsa:4096 -keyout /tmp/privateKey4096.key -out /tmp/certificate4096.crt
    c) Generate checksums for both private keys and certificate files, store in /tmp.

i) E.g. for each file,
$sha256 /tmp/privateKey3072.key >> /tmp/key-sums.txt
$sha256 /tmp/privateKey3072.crt >> /tmp/crt-sums.txt

7) Generate random passphrase for wrapping
   a) Openssl rand -base64 32
8) Shard wrapping passphrase
   a) ssss-split -t 4 -n 8 -w wrappingkey (if greater than 8 shards are created, adjust accordingly)
   b) Create files in /tmp for each shard named “decryption-shard-X”
   c) Copy shards INDIVIDUALLY to USB flash or SD card media
      i) diff input file and output file of each copy
      ii) Label as “shard 1” through “shard x”
      iii) Seal shards in tamper resistant envelopes labeled “shard 1” through X
9) Encrypt private key
   a) Use GPG to encrypt private key
      i) gpg -c -o /tmp/privateKey3072.key.enc /tmp/privateKey3072.key
      ii) gpg -c -o /tmp/privateKey4096.key.enc /tmp/privateKey4096.key
   b) Test shards
      i) ssss-combine -t 4 -n 8
      ii) Compare output of shards 1-4, 3-6, etc. to wrapping key used.

```
Example from shard testing:
$ cat wrapping-key
hbKfBfftB7k7C8mmW+QKA9Tu+zhrZwzZSQi6cOkwe0I=
$ ssss-combine -t3 -n10
Enter 3 shares separated by newlines:
Share [1/3]: wrappingkey-01-3d94cd2ce6e47c84b5a236d47d9f42f610c5c5e6eacead17815f \
fa462c318c0455bbd4eda6d993fb6bf5a177
Share [2/3]: wrappingkey-02-12ac3f004bace5cc96e91bf993c8cfcb05dfc25934b6fd6dd \
32929e9297d4f488d127363eba1adf59d3c0714
Share [3/3]: wrappingkey-03-04f108c1e49bbacc329db733a98a24e7274d9cc1a89a574f59d \
c9957d9e27efa48d29e69af0f42f625b6abce
Resulting secret: hbKfBfftB7k7C8mmW+QKA9Tu+zhrZwzZSQi6cOkwe0I=
```

   c) Test decryption
      i) file /tmp/privateKey3072.key.enc
      ii) file /tmp/privateKey4096.key.enc
      Should return “GPG symmetrically encrypted data (AES cipher)”
      iii) (use output from testing shards above for passphrase for following commands)
      iv) gpg -d /tmp/privateKey3072.key.enc /tmp/privateKey3072.key
      v) gpg -d /tmp/privateKey4096.key.enc /tmp/privateKey4096.key
   d) If test succeeds, we have a properly encrypted key, encrypted with recoverable passphrase from shards requiring a quorum.

10) Copy at minimum one copy each of encrypted private keys and corresponding self signed public key to TWO USB flash or SD card media.
    a) Enclose each in tamper resistant envelopes. Sign and date the envelopes across the flap and body of envelope immediately after sealing.
    b) Each copy will be transported to separate, secure storage facilities to protect against local disaster (fire, flood, etc.). No one who has access to one of the shards should have access to any of the other shards once stored in these secure locations.
11) Place all media generated in a portable safe. The key or combination to this safe should be in the custody of an individual not in possession of the safe.

Checkpoint: At this time, this procedure results in the following:

- Minimum of 2 copies of GPG symmetrically encrypted key material and certificate in tamper resistant envelopes
    - One 3072 RSA key and cert (and checksums?)
    - One 4096 RSA key and cert (and checksums?)
    - 8 SSSS shards of the private key decryption material on USB media sealed in tamper resistant envelopes

Discussion

HSM (Short term) key material generation

This process will create short term copies of encrypted key material, and will also create short term shard material. This material will be used within a short time frame after the key generation in order to load the private key into the AWS CloudHSM. Immediately after this process, the short term key material and shards will be physically destroyed. The process of loading the key into the HSM should also be video recorded.

1) As part of the key generation ceremony, the following steps shall be taken in order to generate the short term key/shards
2) On key generation laptop, generate a random wrapping key.
    a) openssl rand -base64 32 > shortterm.key
    b) cat shortterm.key
3) Shard the key
    a) ssss-split -t 3 -n 3 -w shorttermkey
4) Test the shards
    a) ssss-combine -t3 -n3
5) GPG encrypt the private key using the randomly generated passphrase
    a) gpg -c -o /tmp/ShortTermPrivateKey3072.key.enc /tmp/privateKey3072.key
    b) gpg -c -o /tmp/ShortTermPrivateKey4096.key.enc /tmp/privateKey4096.key
6) Test decryption of the short term private keys

    a) gpg -d /tmp/ShortTermPrivateKey3072.key.enc /tmp/testprivateKey3072.key
    b) gpg -d /tmp/ShortTermPrivateKey4096.key.enc /tmp/testprivateKey4096.key
    c) diff /tmp/testprivateKey3072.key /tmp/privateKey3072.key
    d) diff /tmp/testprivateKey4096.key /tmp/privateKey4096.key
7) Write the shards to individual media, seal and label in tamper resistant envelopes
8) Write the encrypted keys to media, seal and label in individual tamper resistant envelopes.
9) Assuming three individuals are present for key generation ceremony, assign one shard each to these individuals. These same individuals must reconvene soon after with a network connected laptop to load the keys into the AWS CloudHSM. Chain of custody must be maintained.

Final steps

1) Once key generation, storage of encrypted key, shards, and short term key material is complete the following steps shall be performed
    a) Use secure delete (srm) for each relevant file stored in /tmp
    b) Use sfill to wipe free space on /tmp (#sfill /tmp)
    c) Destroy (or securely archive) boot media
    d) Destroy (or securely archive) hardware
    e) Archive a clean copy of all tools used in this process
        i) Clean image of boot media
        ii) Clean image containing secure-delete, SSSS, rng-tools packages
    f) Archive video recording of key generation process with initialed checklists

HSM key loading

Discussion

Loading the key into the HSM at this point involves what is believed to be the highest risk of key compromise at any point in this procedure. It should be performed from a laptop that has as clean of an OS as possible. A bootable ISO may be used if conditions/procedurs allow. The short term decryption key and shards will be used for this process. The intent of the short lived encrypted key and decryption shards is to have a copy that will exist just long enough to be loaded into the HSM and then be physically destroyed immediately afterwards. It does use some of the same safeguards as the long-term storage key material but does not have the same level of quorum requirements because the window for risk is shorter, and it is used for a dedicated purpose and then destroyed. It will still have the protection that no one actor can obtain the plain text copy of the private key.

This process will require network access and an SSH private/public keypair that allows access to a VM in the VPC which has access to the CloudHSM. It will also require HSM “user” credentials to be used. The key generation hardware could be reused for this purpose and potentially could use the same boot media, but networking access will be required. One important note here is the AWS HSM tools will need to be installed on

this laptop, so this should be tested with the OS is intended to be used. AWS only provides packages for Linux x86_64 which is one constraint.

Prerequisites

Production CloudHSM cluster
At least one instance of HSM
HSM is properly set up (accounts, signed CSR, etc.)
Accounts on HSM for key-owner, key-user, and all accounts for HSM quorum participants
Shards for “short lived” key decryption key
“Short lived” encrypted private key

Procedure

1) Boot hardware
2) Install ssss, install AWS tools[14]
3) Install CustomerCA cert used to verify HSM certificate
4) Generate public/private ssh key
5) Create SSH tunnel from key loading laptop to VM in AWS
    a) $ ssh -L 2225:<ip_of_HSM>:2225 ec2-user@<IP_AWS_VM>
    b) For multiple chained tunnels
       $ssh -A -4 -L 2225:127.0.0.1:2225 user@login.internet2.edu
       $ssh -L 2225:<ip_of_hsm>:2225 user@<ip_of_AWS_VM>
6) Configure the CloudHSM software for “127.0.0.1”
7) Restart CloudHSM daemon
   Ubuntu: sudo service cloudhsm-client restart
8) Copy encrypted keys to /tmp/
9) Decrypt keys to load to into HSM
    a) ssss-combine -t3 -n3
    b) gpg -d /tmp/ShortTermPrivateKey3072.key.enc /tmp/privateKey3072.key
    c) gpg -d /tmp/ShortTermPrivateKey4096.key.enc /tmp/privateKey4096.key
10) Log into HSM
    a) loginHSM -u CU -s key-owner -p <password>
    b) enablee2e (enable end-to-end encryption)
11) If no symmetric key exists, create one
    a) genSymKey -t 31 -s 32 -l aes256
12) Import keys
    a) importPrivateKey -f rsa3072.key -l rsa3072-signing -w <handle_of_sym_key>
    b) importPrivateKey -f rsa4096.key -l rsa4096-signing -w <handle_of_sym_key>
13) Share key with key-user
    a) shareKey <key_handle> <user-id> 1
14) Set key-owner password to random value

[14] https://docs.aws.amazon.com/cloudhsm/latest/userguide/install-and-configure-client-linux.html

a) As a CO user or as key-owner
   Openssl rand -base64 20
   changePswd CU key-owner <new password>

15) Set up Quorum authentication[15] on the HSM.
    From AWS guide (this is untested so far)
    Overview of Quorum Authentication

    The following steps summarize the quorum authentication processes. For the specific steps and tools, see Using Quorum Authentication for Crypto Officers.

    Each HSM user creates an asymmetric key for signing. He or she does this outside of the HSM, taking care to protect the key appropriately.

    Each HSM user logs in to the HSM and registers the public part of his or her signing key (the public key) with the HSM.

    When an HSM user wants to do a quorum-controlled operation, he or she logs in to the HSM and gets a quorum token.

    The HSM user gives the quorum token to one or more other HSM users and asks for their approval.

    The other HSM users approve by using their keys to cryptographically sign the quorum token. This occurs outside the HSM.

    When the HSM user has the required number of approvals, he or she logs in to the HSM and gives the quorum token and approvals (signatures) to the HSM.

    The HSM uses the registered public keys of each signer to verify the signatures. If the signatures are valid, the HSM approves the token.

    The HSM user can now do a quorum-controlled operation.

16) The private key is now imported into the HSM, owned by a user with a random password, shared with the key-user for crypto functions, and not exportable by any account other than key-owner. The HSM requires quorum authentication to modify the key owner account and thusly export the private key.
17) Immediately destroy short lived media containing encrypted private key.
18) Media containing shards can be wiped and reused since they are of no value with the encrypted private key having been destroyed
19) Wipe hardware
20) Ensure that no temporary copies of any critical files remain

[15] https://docs.aws.amazon.com/cloudhsm/latest/userguide/quorum-authentication-crypto-officers-first-time-setup.html

Long term storage of key and quorum keys

Discussion

It is assumed there will be more quorum keys for the key decryption key than there will be individuals present at the key generation. Each quorum key should have been placed in a tamper resistant envelope at the key generation ceremony. The custodians for the quorum keys should be selected by this point, and the quorum keys should be distributed to those individuals/locations. Once this distribution is performed, there is no longer an opportunity for a single actor to compromise/copy the key. Chain of custody and proper handling of the material is critical up to this point due to the quorum keys being stored together (e.g. the custodian of the safe and the key/combination to the safe SHALL be separate to this point). Quorum keys should be distributed to their custodians as soon as practical to reduce the amount of time the shards are stored together.

An up-to-date inventory of quorum key custodians must be kept at all times.

By encrypting the long term copies of the stored private key, the storage of the key itself becomes less critical in terms of security. As previously discussed, by encrypting it via AES, recovery of key material from storage is extremely improbable without access to the sharded decryption keys and sufficient quorum to recover the decryption key itself. Therefore, storage for purposes of DR becomes easier, but the process for recovery of the key becomes more complex.

Because the key itself is encrypted, it can be stored in distributed locations without the same level of concern about access as would an unencrypted key. However, because recovery of the key material now depends on a quorum of individuals or shards, planning must account for new and departing individuals in their role as a quorum member, and how the individual shards are stored. I.e. chains of custody for shards must be kept up with and/or periodic resharding of the decryption key must be performed if shards are lost.

Ultimately, the key will be recoverable from a quorum not just of the long term storage of the key, but also by a sufficient quorum on the HSM, so there is some redundancy in DR planning. Where physical and process means were previously used to protect the unencrypted signing key (access to physical safes and safe deposit boxes), this proposed process introduces technical means to enforce a higher quorum threshold.

There exists in terms of physical security three storage locations in Ann Arbor. Two safe deposit boxes, plus one safe in the Internet2 offices. One of these physical storage locations can be used for storing one copy of the encrypted private key. The remaining two may be used for storing two of the shards, one each, of the quorum key. A secure location should be sought outside of Ann Arbor to store one additional copy of the encrypted private key to ensure against local natural disaster.

Depending on the number of shards used for the private key encryption key, and who the custodians of the individual shards are, we will likely have geographically diverse storage of the shards. Physical storage requirements for the shards should be discussed as a group, and we should should come to consensus on

- The number of shards to create
- The quorum threshold required
- Requirements for physical storage of the shards

The shards individually pose little or no risk, since knowledge of a single shard does not compromise the decryption key or the decrypted private key. Similarly, knowledge of the encrypted private key does not compromise the key. Compromise of the key would require compromise of enough shards to reach quorum ***AND*** compromise of the encrypted private key held in secure storage (safe deposit, secure safe, etc.).

Once key generation is complete, the key is loaded in the HSM, and the long term storage is in place, no single actor can compromise the key. Assuming the shards and the quorum private keys for the HSM are stored offline, and in the custody of the quorum members, **a threat actor or natural disaster would need to compromise multiple physical locations in diverse geographic locales to compromise the private key**.

Quorum members

HSM

Required number of Crypto Officers to reach quorum on the HSM shall be no less than three (we may agree on a higher threshold). The total number of Crypto Officers should be no less than five. Initial proposed minimum list of Crypto Officer accounts would include the following staff:

xxxx
xxxx
xxxx
xxxx
xxxx

SSSS

Required number of key recovery shards to reach quorum shall be no less than four. The total number of shards should be no less than six.

Shards shall be stored separate from the encrypted key material, but storage of the shards may be local to the shard custodian. No person shall have access to more than one shard.

Key recovery process

Recovery of key from HSM as part of normal operations and/or vendor migrations

Key recovery should be possible through two methods under normal operations. One of these methods is recovery via quorum mechanisms from the HSM. This should be the first method used in case key recovery is required as part of normal procedures. Some scenarios that could trigger this method could be:

- Migration of the service from AWS CloudHSM to another non-AWS HSM provider
- Migration of the service from AWS to “in-house” hosted service with private hardware HSM
- Migration of the service from AWS CloudHSM to another AWS provided HSM or similar

Because the key_owner account shall have a randomized account password in place, and any account modification operations will require quorum, quorum members should convene (remotely is possible) and agree on the actions to be taken. A designated quorum member will be responsible for changing the key_owner password and recovering the private key. This should be done in the presence of at least one other Internet2 staff member, preferably a quorum member. All actions taken with the key should be observed by this witness. A plan shall be documented in a checklist format, and both persons present shall initial steps and sign completion at the end of the process. This signed document shall be preserved and stored with the primary copy of the encrypted key in Ann Arbor. Key recovery shall take place on a secure laptop via SSH tunnel to AWS.

The designated quorum member for recovery will log into the HSM and generate a quorum request token. Each participating quorum member will generate a token signature and upload to the HSM. Once quorum is achieved, the designated quorum member will then change the password of the key_owner to a known random password and execute the key export (wrapping) command, resulting in a copy of the private key on the secure laptop. This key should then be immediately uploaded to the target where the key is being moved to. Or, if this procedure is being performed to replace a somehow lost long term copy of the encrypted private key, steps taken for generation of the original long-term archive copy should be repeated (gpg-AES encryption, followed by SSSS sharding of the encryption key, secure chain of custody, two person rules, etc.). After the key is either migrated or prepared for long-term storage, any temporary copies of the key shall be securely deleted and steps should be taken to wipe the device the key was temporarily stored on.

Once the procedure is completed, the key should either immediately be deleted from the CloudHSM or the key_owner password should immediately be changed to an unknown random password (which may require another quorum token & signing request).

“Break Glass” Disaster Recovery of key from long-term encrypted storage

In the event that the key is not recoverable from an HSM quorum, and key recovery is needed, the long term copy of the key will need to be decrypted. Quorum shards & custodians are expected to be geographically diverse, and convening in the same physical proximity may be difficult if key recovery is needed in a timely

manner. It is possible to transmit individual shards for this purpose via secure means. PGP, S/MIME, etc. may be used to transmit them to the quorum member or trusted staff member in order to decrypt the private key in Ann Arbor or the alternate location. If encrypted communications are not available, it is conceivable that the key shard could be read over telephone. The process would be tedious, but possible.

One important note is that once the shards are transmitted for this purpose, they should be treated as “use once and dispose of”. Thus, if this process is used, a new decryption key and shards should be created and distributed, and any copies of the key encrypted with the old encryption key should be destroyed and replaced by the newly encrypted key.

If sufficient quorum members are able to convene in one location, then secure transmission of the keys should not be needed since the quorum members would travel and convene with their shards. In this case, the key will not need to be re-encrypted as long as safe handling practices are followed during decryption and needed processing of the signing key. Once the shards are no longer needed for the procedure, they should be sealed in new envelopes as during the key generation process. The use of the shards and their re-sealing shall be noted on the shard inventory.

The procedure for a remote vs. local quorum necessitates creation of unique checklists for each procedure. These SHALL be generated before the long-term key is decrypted. They SHALL be initialed and signed by parties present, and at the end of the procedure stored with the (possibly re-encrypted) long-term storage copy of the key.

Appendix A

Intermediate signing key with root key

We considered the generation of a well protected root key with an intermediate key for metadata signing. After consulting with the FOG list and internal discussions, we decided that the relative risks were greater than any potential benefits and also that there was a possibility of introducing additional complexities and/or incompatibilities. Several SAML packages are known to not be able to handle certificate chaining (SimpleSAMLphp, EZProxy). Additionally, working with an intermediate signing key would require implementation of a Certificate Revocation List (CRL) and would be an additional burden. One possible advantage to working with an intermediate key would have been the potential to generate a non-exportable key on an HSM, but that would result in a mitigated risk being exchanged for a potentially unmitigated risk resulting from a weak or untrustworthy RNG on the HSM.

Appendix B

```
$ rngtest -c 1000  < /dev/random
rngtest 2-unofficial-mt.14
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

rngtest: starting FIPS tests...
rngtest: bits received from input: 20000032
rngtest: FIPS 140-2 successes: 999
rngtest: FIPS 140-2 failures: 1
rngtest: FIPS 140-2(2001-10-10) Monobit: 0
rngtest: FIPS 140-2(2001-10-10) Poker: 0
rngtest: FIPS 140-2(2001-10-10) Runs: 0
rngtest: FIPS 140-2(2001-10-10) Long run: 1
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=250.003; avg=547.754; max=973.690)Kibits/s
rngtest: FIPS tests speed: (min=6.961; avg=16.587; max=16.731)Mibits/s
rngtest: Program run time: 36808087 microseconds


$ dieharder -g 500 -a
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |rands/second|   Seed   |
    /dev/random|  1.71e+04  |1164528802|
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.93388802|  PASSED
      diehard_operm5|   0|   1000000|     100|0.51609103|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.07763580|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.44280146|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.79066745|  PASSED
        diehard_opso|   0|   2097152|     100|0.91780177|  PASSED
        diehard_oqso|   0|   2097152|     100|0.10527227|  PASSED
         diehard_dna|   0|   2097152|     100|0.42775943|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.90920770|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.32119118|  PASSED
```

diehard_parking_lot	0	12000	100	0.08070127	PASSED
diehard_2dsphere	2	8000	100	0.40193690	PASSED
diehard_3dsphere	3	4000	100	0.96568856	PASSED
diehard_squeeze	0	100000	100	0.60884530	PASSED
diehard_sums	0	100	100	0.03881150	PASSED
diehard_runs	0	100000	100	0.00995853	PASSED
diehard_runs	0	100000	100	0.96284723	PASSED
diehard_craps	0	200000	100	0.88976259	PASSED
diehard_craps	0	200000	100	0.97118797	PASSED
marsaglia_tsang_gcd	0	10000000	100	0.77163730	PASSED
marsaglia_tsang_gcd	0	10000000	100	0.89470401	PASSED
sts_monobit	1	100000	100	0.99931463	WEAK
sts_runs	2	100000	100	0.12295404	PASSED
sts_serial	1	100000	100	0.31219921	PASSED
sts_serial	2	100000	100	0.37191104	PASSED
sts_serial	3	100000	100	0.80955868	PASSED
sts_serial	3	100000	100	0.88231087	PASSED
sts_serial	4	100000	100	0.73986298	PASSED
sts_serial	4	100000	100	0.97313573	PASSED
sts_serial	5	100000	100	0.67057736	PASSED
sts_serial	5	100000	100	0.50497161	PASSED
sts_serial	6	100000	100	0.76236629	PASSED
sts_serial	6	100000	100	0.82596953	PASSED
sts_serial	7	100000	100	0.77785762	PASSED
sts_serial	7	100000	100	0.41456334	PASSED
sts_serial	8	100000	100	0.92251841	PASSED
sts_serial	8	100000	100	0.80658950	PASSED
sts_serial	9	100000	100	0.68934699	PASSED
sts_serial	9	100000	100	0.87231313	PASSED
sts_serial	10	100000	100	0.40339034	PASSED
sts_serial	10	100000	100	0.48945967	PASSED
sts_serial	11	100000	100	0.76874130	PASSED
sts_serial	11	100000	100	0.99441787	PASSED
sts_serial	12	100000	100	0.68899950	PASSED
sts_serial	12	100000	100	0.89907501	PASSED
sts_serial	13	100000	100	0.93089816	PASSED
sts_serial	13	100000	100	0.47483595	PASSED
sts_serial	14	100000	100	0.78332524	PASSED
sts_serial	14	100000	100	0.98567669	PASSED
sts_serial	15	100000	100	0.45355767	PASSED
sts_serial	15	100000	100	0.84727819	PASSED
sts_serial	16	100000	100	0.27165006	PASSED
sts_serial	16	100000	100	0.52361363	PASSED

```
          rgb_bitdist|   1|    100000|     100|0.73449197|  PASSED
          rgb_bitdist|   2|    100000|     100|0.82593088|  PASSED
          rgb_bitdist|   3|    100000|     100|0.99068246|  PASSED
          rgb_bitdist|   4|    100000|     100|0.55733146|  PASSED
          rgb_bitdist|   5|    100000|     100|0.78753233|  PASSED
          rgb_bitdist|   6|    100000|     100|0.94905571|  PASSED
          rgb_bitdist|   7|    100000|     100|0.20312057|  PASSED
          rgb_bitdist|   8|    100000|     100|0.64422952|  PASSED
          rgb_bitdist|   9|    100000|     100|0.00893212|  PASSED
          rgb_bitdist|  10|    100000|     100|0.40972938|  PASSED
          rgb_bitdist|  11|    100000|     100|0.53158514|  PASSED
          rgb_bitdist|  12|    100000|     100|0.86961017|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.38935389|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.45974163|  PASSED
rgb_minimum_distance|   4|     10000|    1000|0.98995395|  PASSED
rgb_minimum_distance|   5|     10000|    1000|0.06028821|  PASSED
    rgb_permutations|   2|    100000|     100|0.96288421|  PASSED
    rgb_permutations|   3|    100000|     100|0.68724935|  PASSED
    rgb_permutations|   4|    100000|     100|0.96818092|  PASSED
    rgb_permutations|   5|    100000|     100|0.93407394|  PASSED
      rgb_lagged_sum|   0|   1000000|     100|0.90332926|  PASSED
      rgb_lagged_sum|   1|   1000000|     100|0.23146519|  PASSED
      rgb_lagged_sum|   2|   1000000|     100|0.54658798|  PASSED
      rgb_lagged_sum|   3|   1000000|     100|0.77474592|  PASSED
      rgb_lagged_sum|   4|   1000000|     100|0.39237535|  PASSED
      rgb_lagged_sum|   5|   1000000|     100|0.41384538|  PASSED
… <stopped testing here, had run for 1 week>
```

ToDo:

AI: Shannon to document CO key generation and quorum operation procedures & policies.