REFERENCE

CPU TO TPM BUS PROTECTION GUIDANCE – ACTIVE ATTACK MITIGATIONS

Version 1.0
Revision 30
May 8, 2023

Contact: admin@trustedcomputinggroup.org

Draft

DISCLAIMERS, NOTICES, AND LICENSE TERMS

CHANGE HISTORY

REVISION	DATE	DESCRIPTION
V1.0 R30	May 8, 2023	Initial Publication

CONTENTS

Contributors

TCG would like to thank the following contributors to this document.

Robert Strong	Advanced Micro Devices, Inc.
Alex Tzonkov	Advanced Micro Devices, Inc.
Michael Zhang	Advanced Micro Devices, Inc.
Jen Ye	Advanced Micro Devices, Inc.
Frederick Otumfuor	AMI
Stuart Yoder	ARM Ltd.
Monty Wiseman	Beyond Identity
Scott Phuong	Cisco
Marc Beamon	Dell, Inc.
Travis Gilbert	Dell, Inc.
Amy C Nelson	Dell, Inc.
Jason Kolodziej	Dell, Inc.
Michael Eckel	Fraunhofer Institute for Secure Information Technology (SIT)
Henk Birkholz	Fraunhofer Institute for Secure Information Technology (SIT)
Jeff Anderson	Google Inc.
Chris Fenner	Google Inc.
Shiva Dasari	Hewlett Packard Enterprise
Theo Koulouris	Hewlett Packard Enterprise
Vali Ali	HP Inc.
David Roderick	HP Inc.
Adrian Shaw	HP Inc.
Joshua Schiffman	HP Inc.
Silviu Vlasceanu	Huawei Technologies Co., Ltd.
Ken Goldman	IBM
Joerg Borchert	Infineon Technologies
Ga-Wai Chin	Infineon Technologies
Andreas Fuchs	Infineon Technologies
Guillaume Raimbault	Infineon Technologies
Sven Schuch	Infineon Technologies
Florian Schweiger	Infineon Technologies
Imran Desai	Intel Corporation
Ned Smith	Intel Corporation
Liran Perez	Intel Corporation
Jiewen Yao	Intel Corporation
Robert Hart	Johns Hopkins University Applied Physics Lab
Eric Sivertson	Lattice Semiconductor Corp.
Bill Kweon	Lenovo (United States) INC
Masoud Manoo	Lenovo (United States) INC
Ronald Aigner	Microsoft
Lonnie Harrell	Microsoft
Brad Litterell	Microsoft
Erich McMillan	Microsoft
Dana Amsalem Cohen	Nuvoton Technology Corporation
Dan Morav	Nuvoton Technology Corporation
Ludovic Jacquin	Nvidia Corporation
Joe Pennisi	Nvidia Corporation
Dick Wilkins	Phoenix Technologies

Olivier Collart	STMicroelectronics
Nourdine El Idrissi	STMicroelectronics
Yves Magnaud	STMicroelectronics
Charley Villette	STMicroelectronics
Zachary Blum	United States Government
Lawrence Reinert	United States Government

1 SCOPE

This reference document describes a method to protect communication between a CPU and a TPM connected by an unprotected bus from active attacks. The transmission of an object to and from the TPM across the physical interface can be done securely in many cases, but the means to do so may not be obvious from existing specifications. The capabilities of the TPM can protect against passive attacks, but additional solutions are necessary to protect against active attacks. This document describes one solution for protecting against active attacks and is by no means exhaustive. The document includes prototype scripts to enable prototyping.

This document assumes knowledge of the TPM Library Specification [1].

2 Problem statement

If a TPM device and an attacker-controlled device are connected to the same plain-text communication path, the attacker can leverage active attacks to intercept and modify any traffic that is exchanged between the CPU and the TPM device.

The attacker can use active techniques learn secrets that can then be used to reveal data on the disk or to gain access to network resources.

Example 1: There is an embedded controller (EC) controlling the bus traffic on the SPI where the TPM is connected. If the EC can be exploited by an attacker, the attacker can install software on the EC that sniffs the traffic to the TPM. If the EC cannot be exploited, it could be replaced with an EC that is connected to both its own Chip Select (CS) and the TPM CS. The EC can then either record a whole boot sequence or just the unsealed secret if the TPM returns that in plain text.

Example 2: The attacker, with knowledge of the measurements extended to the TPM during boot via the TCG Event Log, moves the TPM to a system the attacker controls and replays the extend operations to recreate PCR values that will unseal the secret.

3 Protecting a sealed secret using an NV Index

A common use case for the TPM involves protection of an object, e.g., a symmetric encryption key. This use case is accomplished by encrypting the object with a key held only inside of the TPM's protected boundary. Additionally, the protected object can be associated with TPM capabilities such as enhanced authorization, e.g., a password policy or evidence of the platform configuration as represented in the TPM's Platform Configuration Registers, PCR.

The software layer of the host platform conveys the object to the TPM via the host CPU, where it is transferred via a physical interface, such as an SPI bus, to the TPM from the chipset.

The design described in Section 3.3 is one example of using a secret, the Host Secret. The Host Secret is stored in protected platform storage to enable the protected transfer of an object to and from the TPM. Depending on the threat model, the protected platform storage might be in the CPU or some other component with non-volatile storage that only the host firmware can access. This design also requires on use of an NV Index configured with the TPM_NT_EXTEND attribute.

One possible implementation of this design involves platform firmware extending the Host Secret and application software making use of the NV index to protect application secrets.

3.1 Goal

The objective of this design is to seal a secret to a TPM, such that the secret can only be recovered on the same host platform under the exact same conditions that were present when sealing the secret.

This design protects against the following active attack:

1. An attacker with knowledge of the communication to the TPM moves the TPM to another platform, to recover the secret.
    a. The mitigation for this attack is to utilize a Host Secret known only to the original host.

3.2 Assumptions

This design assumes that the host platform has a unique Host Secret, e.g. provisioned in the CPU. The Host Secret is unique to individual platforms. Further, the Host Secret is only accessible to trusted software. Trusted software can be provisioning tools that run in a controlled environment or host platform firmware. The Host Secret cannot be migrated to a different host platform.

This design requires an asymmetric key with a certificate and certificate chain. The diagrams, text, and code samples in Section 3.103.9 use the Endorsement Key (EK), EK certificate [2], and respective certificate chain as the asymmetric key and its certificate chain.

This design requires the platform to be provisioned with the entire certificate chain from the TPM vendor's CA certificate to the EK certificate and have the capability in host platform firmware and host software to validate the EK certificate chain.

The examples in Section 3.10 are simplified to use a single Host Secret. Depending on the threat model, this might be inadequate. In some cases, it might be necessary to have more than one Host Secret. The examples assume different use cases, which results in different approaches to provisioning. The first example assumes a single provisioner who provisions both the NV Index and the Sealed secret, e.g., a platform OEM sealing a platform secret to the TPM. The second example assumes different provisioners for the NV Index, e.g., the Platform OEM, and the Sealed Secret, e.g., the Host OS.

3.3 Design

This design uses the Host Secret in two ways:

1. As a seed to encrypt traffic over the TPM bus.
2. As authorization to recover the sealed secret.

This design uses a verified EK to establish a Host Secret that is shared between the CPU and TPM.

The method by which a Host Secret is derived may vary between platforms. The TCG DICE Layering Architecture Specification [3] defines one possible solution for the creation or derivation of a Host Secret in Section 6.2. Another solution is to use a KDF.

Although the sample scripts in Section 3.10 use the Host Secret directly, host software should actually derive two values from the Host Secret (see Section 3.6 Usage in Early Boot), one for use as the NV Index's password, another for use as the value extended into the NV Extend Index.

The implementation of the steps described below may vary between platforms.

3.4 Provisioning the NV Index

Trusted software, for instance platform firmware, can detect the absence of the NV Index and provision the NV Index. This step can be done either in a secure manufacturing environment or by trusted software post manufacturing.

1. Create an EK and a primary storage key if they are not already provisioned. It is unnecessary to persist the keys. Validate the EK against the EK certificate chain to ensure that a MIM has not provided a counterfeit EK. See Figure 1 Initial Provisioning of the Sealed Secret, Steps 1-3, and Figure 4 Initial Provisioning Steps 2-4.
2. Create a salted session with the TPM using the EK and use that salted session for all remaining steps in Section 3.4. See Figure 1 Initial Provisioning of the Sealed Secret Step 4 and Figure 4 Initial Provisioning Step 7.

The salted session HMAC ensures that the primary storage key Name is authentic. It also encrypts the primary storage key password, although this is not essential.

Using the salted session created in Section 3.4 the NV Index is provisioned as part of the provisioning steps.

3. Create an NV Index with a TPM_NT_EXTEND attribute. See Figure 1 Initial Provisioning of the Sealed Secret Steps 4 and 5 and Figure 4 Initial Provisioning Step 8.

Use the Host Secret as the password for the NV Extend Index so the password can later be used for a bind session. The password for the NV Extend Index is not used for authorization, only for the bind session. Set the following NV Index attributes:

- Extend – because, by definition, this is an NV Extend index.
- Orderly – to prevent NV wear out.
- CLEAR_STCLEAR – to enforce reset of the NV Index on a TPM Reset or TPM Restart.
- Platform Create – to ensure the NV Index is preserved after a TPM2_Clear.
- Policy Write – to enable the use case.
- Policy Read – to enable the use case.
- noDA – this might be desired, depending on the policy, but is not used in this example.

The salted session, created in Step 2, provides confidentiality for the Host Secret as it is transmitted to the TPM to create the NV Index. The salted session's HMAC ensures that the NV Index Name is authentic.

The policy for the NV Index is the OR of elements A | B | C as follows:

A - Command code NV Read - anyone can read the index.

B - Command code NV extend - anyone can extend the index.

C - Command code Policy NV - anyone can use the index in a TPM2_PolicyNV command.

3.5 Sealing the application secret

This step may be done as part of provisioning or may be done post provisioning.

Using the salted session created in Section 3.4 the application secret is sealed as part of the provisioning steps.

1. Generate any secret(s) that an application wishes to protect from interception on the bus between CPU and TPM. See Figure 6 Sealing the Secret Step 5.

Use TPM2_Create to seal the secret to the primary storage key.

2. Run the TPM2_Create command in the salted session from Step 2 of Section 3.4 using the EK in a parameter-decryption session to encrypt the sealed secret when it is transmitted over the bus. See Figure 6 Sealing the Secret Steps 6-9.

The policy for the sealed data is the AND of the following elements:

- D: TPM2_PolicyNV - where the data must be equal to the value in the NV Index, which resulted from extending the Host Secret.
- E: TPM2_PolicyCommandCode with TPM_CC_Unseal – to only allow unsealing of the data.

To create the policy for the sealed data, first calculate the expected NV Index value (the result of the extend of the Host Secret into the NV Index). Then, perform the calculation for the TPM2_PolicyNV command with the expected NV index value as just calculated. The calculation can be done without using the TPM.

3.6 Usage in Early Boot

Early platform firmware needs to perform the following step on every boot when it has access to the Host Secret.

1. Start a new authorization session with the TPM, binding to the NV Index. See Figure 2 Extending the Host Secret Step 4 and Figure 5 Fulfilling policy for the Seal/UnSeal process Step 4.

The bind permits parameter encryption based on the Host Secret. A salted session could be used, but salting requires a slower asymmetric key operation, while bind uses a hash operation.

2. Using Policy component B, extend the Host Secret into the NV Index. Use command parameter encryption based on the bind session to provide confidentiality for the Host Secret on the bus. See Figure 2 Extending the Host Secret Steps 5-7 and Figure 5 Fulfilling policy for the Seal/UnSeal process Step 5.

As mentioned in Section 3.3 Design, the Host Secret is the basis for both the key used for parameter encryption (1.) and the value extended into the NV Index (2.). A safer approach is to derive both from the same Host Secret but be different values, using a KDF with two different hardcoded inputs, e.g., “extended secret string” and “bind secret string”.

3.7 Unsealing the application secret

This step does not require access to the Host Secret and can be performed by any software on the platform.

If the storage parent was not persisted, recreate it. Then:

1. Load the sealed data. No authorization is required.

2. Start one policy session for the NV Index Policy NV requirement.
3. Satisfy Policy component C using TPM2_PolicyCommandCode with TPM_CC_PolicyNV.
4. Start a second policy session for the unseal. This session must be a salted session using the EK for the response parameter (the sealed secret) encryption.
5. Satisfy the second policy using TPM2_PolicyCommandCode with TPM_CC_Unseal AND TPM2_PolicyNV equal to extending the Host Secret into the NV index.

The TPM2_PolicyNV command uses the first session for NV Index authorization. TPM2_PolicyNV requires the correct value in the NV Index. That value is not secret and can in fact be read from the NV Index for debugging. However, only the platform firmware on the platform that has access to the Host Secret can extend the NV Index to create the correct value.

If the TPM is subsequently moved, the new platform does not know the Host Secret, and therefore cannot reproduce the NV index data. That is, an attacker knows the correct digest that should be in the NV Index, but cannot reproduce it.

6. Run the unseal using the second policy session for authorization. The salted session encrypts the response parameter, which is the application secret in plain text. See Figure 3 Unsealing the Secret Steps 8-11 and Figure 7 Unsealing the secret Steps 7-11.

3.8 Summary

The TPM traffic is confidentiality and integrity protected. If an attacker moves the TPM, it cannot unseal the protected secret because it cannot recreate the NV extend index value without knowing the Host Secret.

3.9 References

[1] TCG TPM Library Specification Family 2.0, Revision 1.59 or later

[2] TCG EK Credential Profile for TPM Family 2.0, Version 2.4 or later

[3] DICE Layering Architecture Specification, Version 1.0, Revision 0.19 or later

[4] TCG TSS 2.0 Overview and Common Structures Specification, Version 0.90, Revision 03 or later

[5] IBM TSS and tools: https://sourceforge.net/projects/ibmtpm20tss/files/

3.10 Sample script

3.10.1 Sequence Diagrams for Sample Scripts using ibmtpm20tss

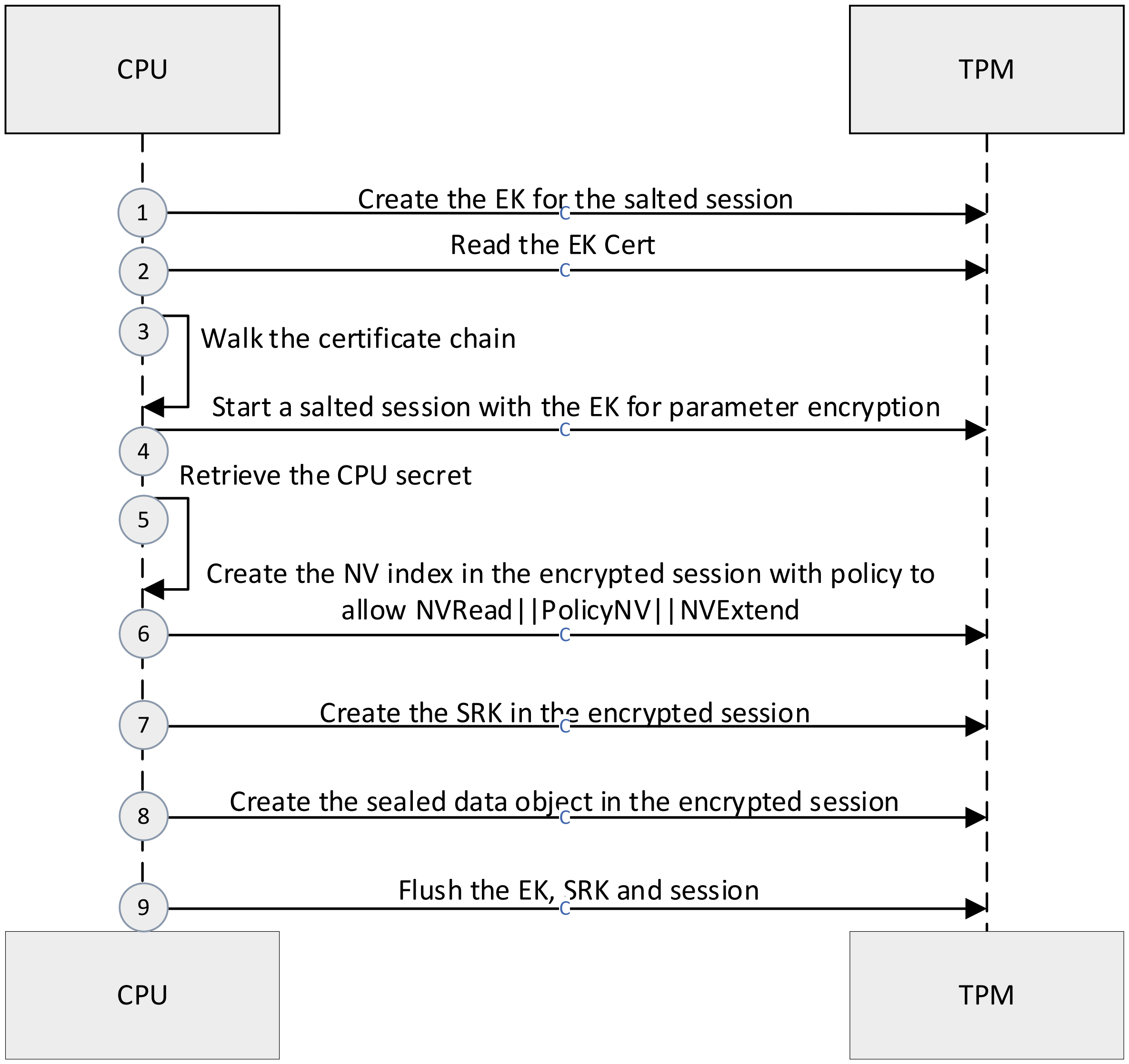
CPU
TPM
1
Create the EK for the salted session
2
Read the EK Cert
3
Walk the certificate chain
Start a salted session with the EK for parameter encryption
4
Retrieve the CPU secret
5
Create the NV index in the encrypted session with policy to allow NVRead||PolicyNV||NVExtend
6
7
Create the SRK in the encrypted session
8
Create the sealed data object in the encrypted session
9
Flush the EK, SRK and session
CPU
TPM

Figure 1 Initial Provisioning of the Sealed Secret

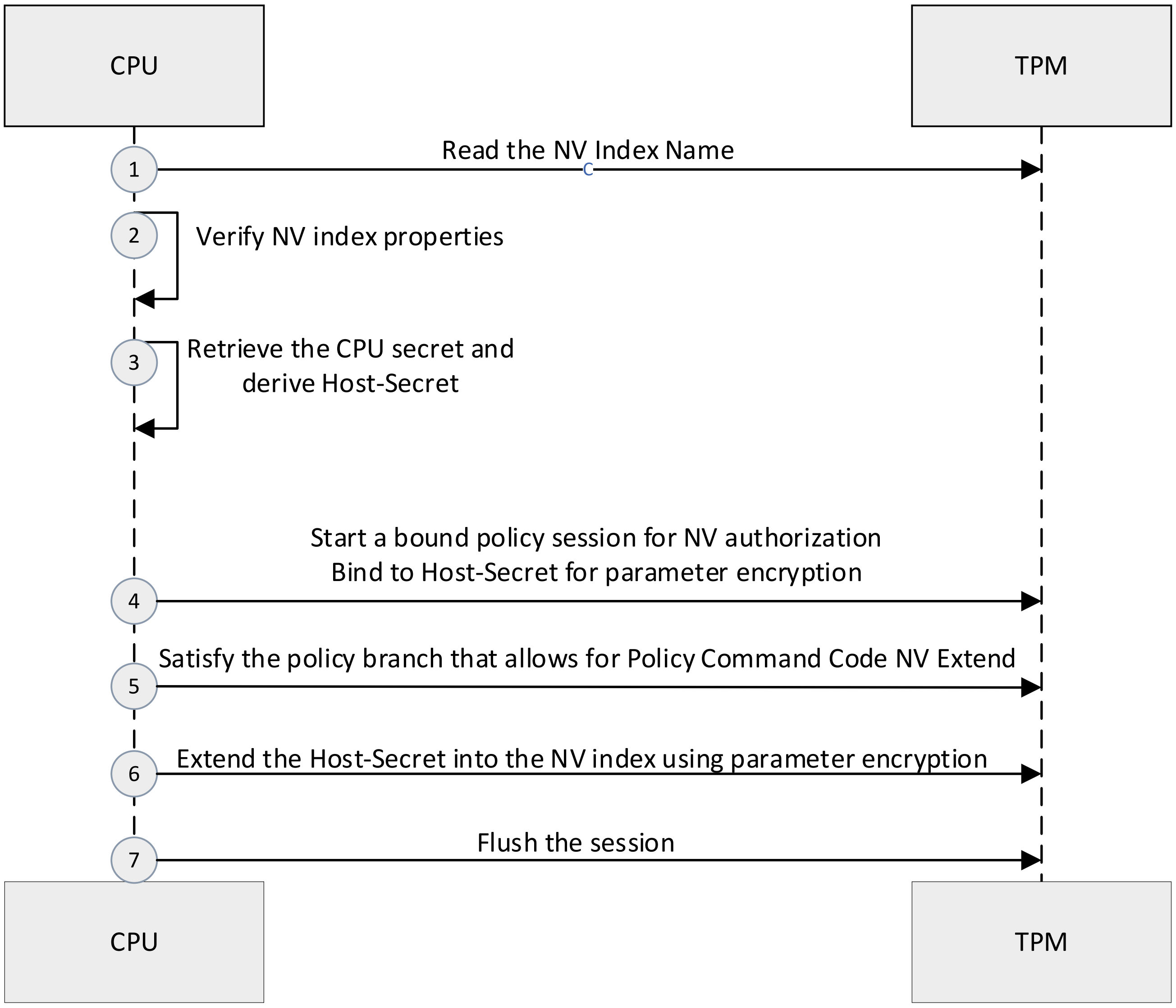
CPU
TPM
1
Read the NV Index Name
2
Verify NV index properties
3
Retrieve the CPU secret and derive Host-Secret
4
Start a bound policy session for NV authorization
Bind to Host-Secret for parameter encryption
5
Satisfy the policy branch that allows for Policy Command Code NV Extend
6
Extend the Host-Secret into the NV index using parameter encryption
7
Flush the session
CPU
TPM

Figure 2 Extending the Host Secret

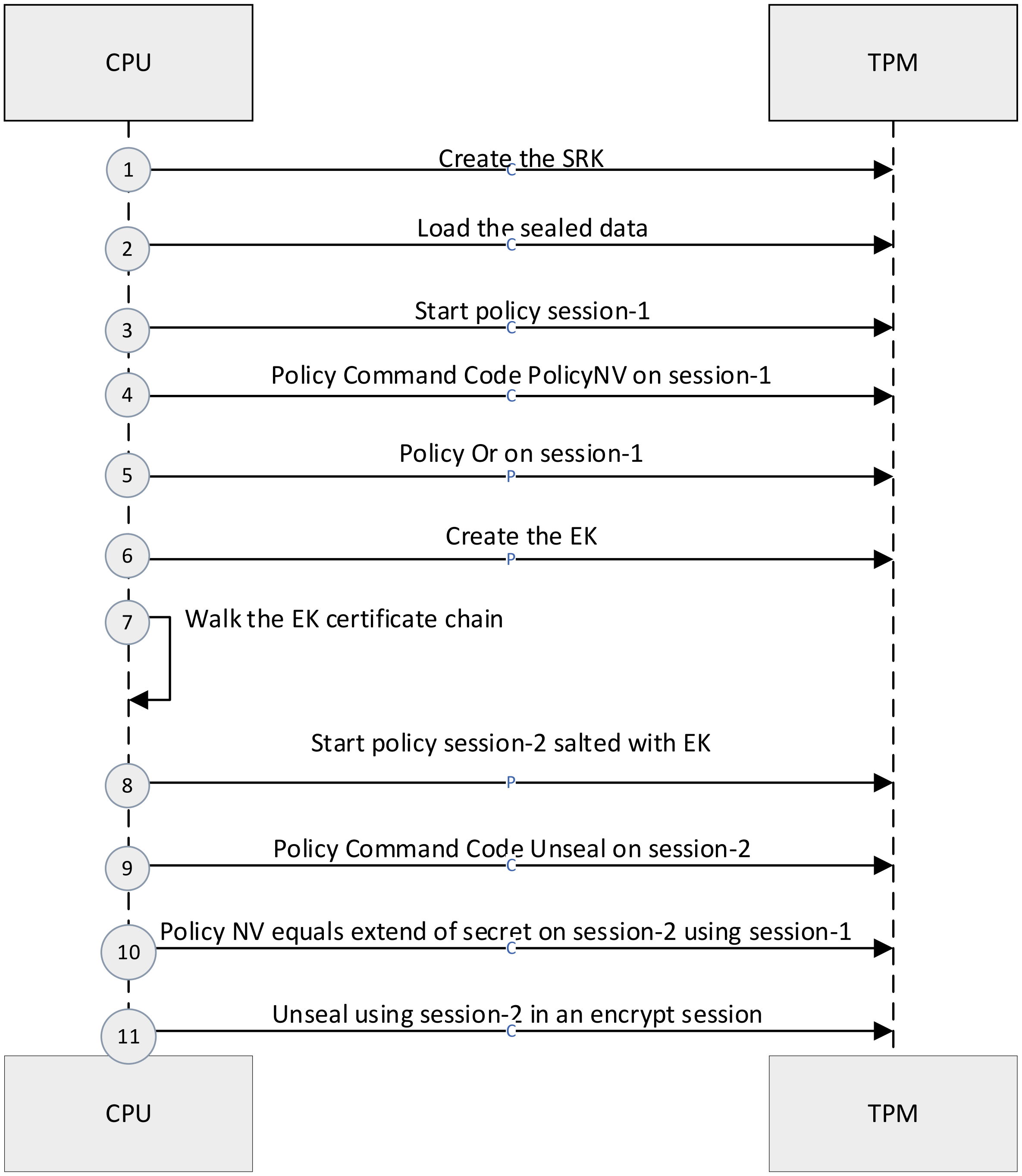
CPU
TPM
1
Create the SRK
C
2
Load the sealed data
C
3
Start policy session-1
C
4
Policy Command Code PolicyNV on session-1
C
5
Policy Or on session-1
P
6
Create the EK
P
7
Walk the EK certificate chain
8
Start policy session-2 salted with EK
P
9
Policy Command Code Unseal on session-2
C
10
Policy NV equals extend of secret on session-2 using session-1
C
11
Unseal using session-2 in an encrypt session
C
CPU
TPM

Figure 3 Unsealing the Secret

3.10.2 Implementation using ibmtpm20tss

```
#!/bin/bash
# This script requires the TSS and command line utilities provided by IBM at:
# https://sourceforge.net/projects/ibmtpm20tss/files/

# for rapid prototyping with scripts
export TPM_ENCRYPT_SESSIONS=0
export TPM_DATA_DIR=.

PREFIX=./

checkSuccess()
{
    if [ $1 -ne 0 ]; then
    echo " ERROR:"
    cat run.out
    exit 255
    else
    echo " INFO:"
    fi
}

checkFailure()
{
    if [ $1 -eq 0 ]; then
    echo " ERROR:"
    cat run.out
    exit 255
    else
    echo " INFO:"
    fi
}

# Above here is boilerplate for all scripts

# Temporary files

# tmpnvapol.bin - NV Index policy A
# tmpnvbpol.bin - NV Index policy B
# tmpnvcpol.bin - NV Index policy C
# tmpnvpol.bin - NV Index policy
# tmpext.bin - extend of Host Secret
# tmpuspol.bin - Unseal data policy
# tmpseal.txt - test sealed data in plaintext
# tmppub.bin - sealed data public part
# tmppriv.bin - sealed data private part
# tmpunseal.txt - unsealed data in plaintext

# Notation

# DEMO: Actions that are just for the demo script, not for the actual application
# PROVISION: Actions taken once during provisioning
# RUNTIME: Actions taken at runtime

# Policies

# NV Extend Index
# Policy OR of
# Policy A - command code NV read
# Policy B - command code NV extend
# Policy C - command code Policy NV

# Sealed Data
# Policy command code unseal
# AND
# policynv equals the Host Secret extended into the NV Index

# If this changes, the unseal policynv calculation must also change
CPU_SECRET=cpusecret

# test sealed secret
echo "sealedsecret" > tmpseal.txt

#
# basic TPM power up, startup, create primary key
#

# clean up from previous run
${PREFIX}nvundefinespace -ha 01000000 -hi  p > run.out

echo "DEMO: power cycle"
${PREFIX}powerup > run.out
checkSuccess $?

echo "DEMO: startup"
${PREFIX}startup > run.out
checkSuccess $?
```

```
echo "DEMO: Create EK certificate for SW TPM"
${PREFIX}createekcert -rsa 2048     -cakey cakey.pem    -capwd rrrr > run.out
checkSuccess $?

#
# Provision the NV Extend Index
#

# The NV index policy permits unauthorized read OR extend

# Policy A - command code NV read
# tmp.txt:
# 0000016c0000014e
# > policymaker -if tmp.txt -ns -v -of tmpnvapol.bin
# 47ce3032d8bad1f3089cb0c09088de43501491d460402b90cd1b7fc0b68ca92f

# Policy B - command code NV extend
# 0000016c00000136
# > policymaker -if tmp.txt -ns -v -of tmpnvbpol.bin
# b6a2e7142ee56fd978047488483daa5b42b8dc4cc7ddcceddfb91793cf1ff1b7

# Policy C - command code Policy NV
# 0000016c00000149
# > policymaker -if tmp.txt -ns -v -of tmpnvcpol.bin
# 203e4bd5d0448c9615cc13fa18e8d39222441cc40204d99a77262068dbd55a43

# policyor
# tmp.txt:  policy OR command code | Policy A | Policy B | Policy C
#
0000017147ce3032d8bad1f3089cb0c09088de43501491d460402b90cd1b7fc0b68ca92fb6a2e7142ee56fd9780474884
83daa5b42b8dc4cc7ddcceddfb91793cf1ff1b7203e4bd5d0448c9615cc13fa18e8d39222441cc40204d99a77262068db
d55a43
# > policymaker -if tmp.txt -ns -v -of tmpnvpol.bin
# 7f17937e206279a3f755fb60f40cf126b70e5b1d9bf202866d527613874a64ac

echo ""
echo "Provision NV Index and Create Sealed Blob"
echo ""

# createek also validates the EK public key against the EK certificate and
# walks the certificate chain.  It leaves the EK loaded at 80000000

echo "PROVISION: Create the EK for the salted session 80000000"
${PREFIX}createek -rsa 2048 -cp -noflush -root certificates/rootcerts.txt > run.out
```

```
checkSuccess $?

echo "PROVISION: Start the EK salted session 02000000 for an authenticated channel"
${PREFIX}startauthsession -se h -hs 80000000 > run.out
checkSuccess $?

# the salted session HMAC ensures that the storge key Name is authentic

echo "PROVISION: Create the primary parent for the unseal data 80000001"
${PREFIX}createprimary -hi p -pwdk sto -se0 02000000 21 > run.out
checkSuccess $?

# the salted session encrypts the NV Index password, the Host Secret,
# and ensures that the NV Index Name is authentic

echo "PROVISION: Define the NV Index, use an encrypt session to encrypt the password"
${PREFIX}nvdefinespace -ha 01000000 -hi p -pwdn ${CPU_SECRET} -ty e +at ody +at stc -pol
tmpnvpol.bin -se0 02000000 21 > run.out
checkSuccess $?

echo "DEMO: NV Extend to set the written bit, needed for the NV Index Name"
${PREFIX}nvextend -ha 01000000 -ic 0 -pwdn ${CPU_SECRET} > run.out
checkSuccess $?

echo "DEMO: Read the NV Index Name"
${PREFIX}nvreadpublic -ha 01000000 -ns > run.out
checkSuccess $?

# NV Index Name 000bbc2784f51dda6d27b92784068c6b8c7c94a4cc530b434e16ef95222fe68e6c92

# Calculate the hash of the Host Secret for the unseal.  Use
# policymaker to calculate the eventual NV extend result in software.

# 'cpusecret' in hexascii
# tmp.txt:
# 637075736563726574
# > policymaker -if tmp.txt -ns -of tmpext.bin
# policy digest:
# 0ad80f8e4450587760d9137df41c9374f657bafa621fe37d4d5c8cecf0bcce5e

# Calculate the sealed object policy
# Policy command code unseal AND policynv equals

# AND term 1 command code unseal
# 0000016c0000015e

# AND term 2 policynv

# args = Hash of operandB.buffer || offset || operation)

# tmp.txt is args input in hexascii
# 0ad80f8e4450587760d9137df41c9374f657bafa621fe37d4d5c8cecf0bcce5e00000000

# Use policymaker with -nz to do a hash of hexascii
# > policymaker -nz -if tmp.txt -v -ns
# args is a hash of the above input:
# 19936a82d9b3fabcc3794b1b9c1dbb71a7de7f6e360cb01f6a6f082f7e66dc60

# CC_PolicyNV || args || Name
#
0000014919936a82d9b3fabcc3794b1b9c1dbb71a7de7f6e360cb01f6a6f082f7e66dc60000bbc2784f51dda6d27b9278
4068c6b8c7c94a4cc530b434e16ef95222fe68e6c92

# Combine the two AND terms to calculate the policy
# tmp.txt
# 0000016c0000015e
#
0000014919936a82d9b3fabcc3794b1b9c1dbb71a7de7f6e360cb01f6a6f082f7e66dc60000bbc2784f51dda6d27b9278
4068c6b8c7c94a4cc530b434e16ef95222fe68e6c92

# > policymaker -if tmp.txt -ns -v -of tmpuspol.bin
#  intermediate policy digest length 32
#  e6 13 13 70 76 52 4b de 48 75 33 86 58 84 e9 73
#  2e be e3 aa cb 09 5d 94 a6 de 49 2e c0 6c 46 fa
#  intermediate policy digest length 32
#  b2 f6 13 21 27 36 b6 f1 c2 84 07 a3 fb a2 7e 14
#  c1 84 c8 21 34 3a 8c 3b fe 23 cd 5f 2e 76 d0 51
# policy digest:
# b2f613212736b6f1c28407a3fba27e14c184c821343a8c3bfe23cd5f2e76d051

echo "PROVISION: Create the sealed data object under the primary key 80000000, encrypt session"
${PREFIX}create -hp 80000001 -pwdp sto -bl -if tmpseal.txt -kt f -kt p -pol tmpuspol.bin -uwa -
opu tmppub.bin -opr tmppriv.bin -se0 02000000 20 > run.out
checkSuccess $?

echo "PROVISION: Flush the EK"
${PREFIX}flushcontext -ha 80000000 > run.out
checkSuccess $?
```

```
#
# Simulate a power cycle to clear the NV extend index
#
echo "DEMO: Power cycle"
${PREFIX}powerup > run.out
checkSuccess $?

echo "DEMO: Startup"
${PREFIX}startup > run.out
checkSuccess $?

echo ""
echo "Usage - Extend the Host Secret unto the NV Index"
echo ""

#
# Usage - Extend the Host Secret unto the NV Index
#

# Real code would read the NV Index Name at reboot and validate the
# value, to ensure that the NV Index has not been undefined and then
# defined differently.

echo "RUNTIME: Read the NV Index Name"
${PREFIX}nvreadpublic -ha 01000000 -ns > run.out
checkSuccess $?

echo "RUNTIME: Start policy session 03000000 for NV authorization, bind to Host Secret for
parameter encryption"
${PREFIX}startauthsession -se p -bi 01000000 -pwdb ${CPU_SECRET} > run.out
checkSuccess $?

echo "RUNTIME: Policy command code NV extend"
${PREFIX}policycommandcode -ha 03000000 -cc 00000136 > run.out
checkSuccess $?

echo "DEMO: Should be policy B first intermediate value b6a2 ..."
${PREFIX}policygetdigest -ha 03000000 > run.out
checkSuccess $?

echo "RUNTIME: Policy OR the NV Policies A, B, C"
${PREFIX}policyor -ha 03000000 -if tmpnvapol.bin -if tmpnvbpol.bin -if tmpnvcpol.bin > run.out
checkSuccess $?

echo "DEMO: Should be policy OR 7f17 ..."
```

```
${PREFIX}policygetdigest -ha 03000000 > run.out
checkSuccess $?

echo "RUNTIME: Extend the Host Secret into the NV Index, use parameter encryption"
${PREFIX}nvextend -ha 01000000 -ic ${CPU_SECRET} -se0 03000000 21 > run.out
checkSuccess $?

echo "DEMO: Policy restart, set back to zero"
${PREFIX}policyrestart -ha 03000000 > run.out
checkSuccess $?

echo "DEMO: Policy command code NV read"
${PREFIX}policycommandcode -ha 03000000 -cc 0000014e > run.out
checkSuccess $?

echo "DEMO: Policy OR"
${PREFIX}policyor -ha 03000000 -if tmpnvapol.bin -if tmpnvbpol.bin -if tmpnvcpol.bin > run.out
checkSuccess $?

echo "DEMO: Read NV Index, should be extend of Host Secret 0ad8 ..."
${PREFIX}nvread -ha 01000000 -se0 03000000 0 > run.out
checkSuccess $?

echo ""
echo "Usage - Unseal"
echo ""

echo "RUNTIME: Create the primary parent for the unseal data 80000000"
${PREFIX}createprimary -hi p -pwdk sto > run.out
checkSuccess $?

echo "RUNTIME: Load the sealed data 80000001"
${PREFIX}load -hp 80000000 -pwdp sto -ipu tmppub.bin -ipr tmppriv.bin > run.out
checkSuccess $?

echo "RUNTIME: Start a PolicyNV authorization policy session 03000000"
${PREFIX}startauthsession -se p > run.out
checkSuccess $?

echo "RUNTIME: Policy command code PolicyNV"
${PREFIX}policycommandcode -ha 03000000 -cc 00000149 > run.out
checkSuccess $?

echo "RUNTIME: Policy OR the NV Policies A, B, C"
${PREFIX}policyor -ha 03000000 -if tmpnvapol.bin -if tmpnvbpol.bin -if tmpnvcpol.bin > run.out
```

```
checkSuccess $?

echo "RUNTIME: Create the EK for the salted session 80000002"
${PREFIX}createek -rsa 2048 -cp -noflush -root certificates/rootcerts.txt > run.out
checkSuccess $?

echo "RUNTIME: Start a unseal policy session 03000001, salt for response parameter encryption"
${PREFIX}startauthsession -se p -hs 80000002 -pwdb ${CPU_SECRET} > run.out
checkSuccess $?

echo "RUNTIME: Policy command code Unseal"
${PREFIX}policycommandcode -ha 03000001 -cc 0000015e > run.out
checkSuccess $?

echo "DEMO: Should be policy Unseal first intermediate value e6 13 13 70 ..."
${PREFIX}policygetdigest -ha 03000001 > run.out
checkSuccess $?

echo "RUNTIME: Policy NV, operation equals extend of Host Secret"
${PREFIX}policynv -ha 01000000 -hs 03000001 -op 0 -if tmpext.bin -se0 03000000 0 > run.out
checkSuccess $?

echo "DEMO: Should be policy Unseal second intermediate value b2 f6 13 21 ..."
${PREFIX}policygetdigest -ha 03000001 > run.out
checkSuccess $?

echo "RUNTIME: Unseal, use the bind encrypt session"
${PREFIX}unseal -ha 80000001 -of tmpunseal.txt -se0 03000001 40
checkSuccess $?

echo "DEMO: Verify the unseal result"
diff tmpseal.txt tmpunseal.txt  > run.out
checkSuccess $?

# cleanup

echo "DEMO: Undefine the NV Index"
${PREFIX}nvundefinespace -ha 01000000 -hi  p > run.out
checkSuccess $?

echo "DEMO: Flush sealed data"
${PREFIX}flushcontext -ha 80000001 > run.out
checkSuccess $?

echo "DEMO: Flush primary storage key"
```

```
${PREFIX}flushcontext -ha 80000000 > run.out
checkSuccess $?

rm -f tmpseal.txt
rm -f tmpunseal.txt
rm -f run.out
rm -f tmppriv.bin
rm -f tmppub.bin
exit
```

3.10.3 Sequence diagrams using tpm2-software

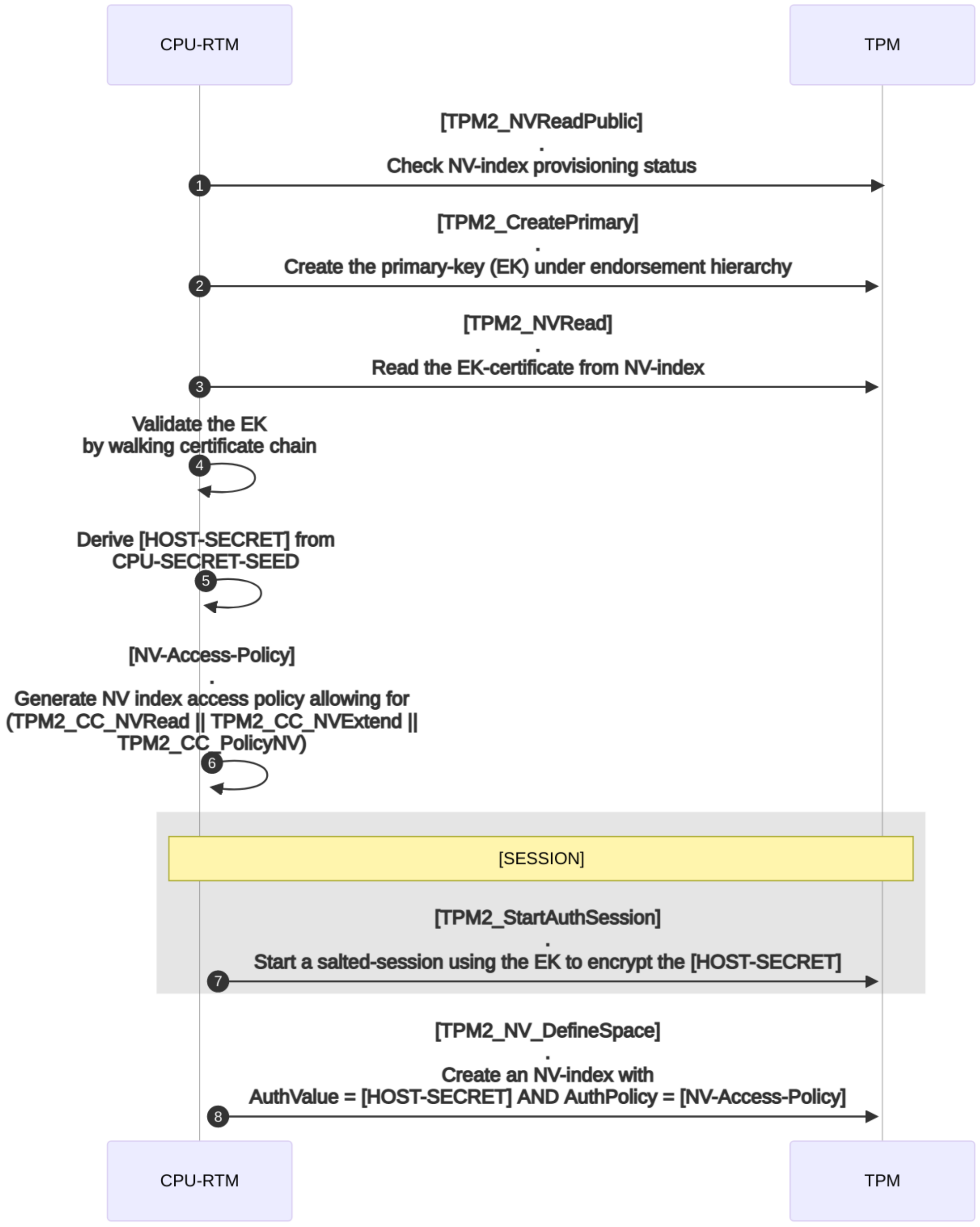
CPU-RTM
TPM
[TPM2_NVReadPublic]
.
Check NV-index provisioning status
1
[TPM2_CreatePrimary]
.
Create the primary-key (EK) under endorsement hierarchy
2
[TPM2_NVRead]
.
Read the EK-certificate from NV-index
3
Validate the EK
by walking certificate chain
4
Derive [HOST-SECRET] from
CPU-SECRET-SEED
5
[NV-Access-Policy]
.
Generate NV index access policy allowing for
(TPM2_CC_NVRead || TPM2_CC_NVExtend ||
TPM2_CC_PolicyNV)
6
[SESSION]
[TPM2_StartAuthSession]
.
Start a salted-session using the EK to encrypt the [HOST-SECRET]
7
[TPM2_NV_DefineSpace]
.
Create an NV-index with
AuthValue = [HOST-SECRET] AND AuthPolicy = [NV-Access-Policy]
8
CPU-RTM
TPM

Figure 4 Initial Provisioning

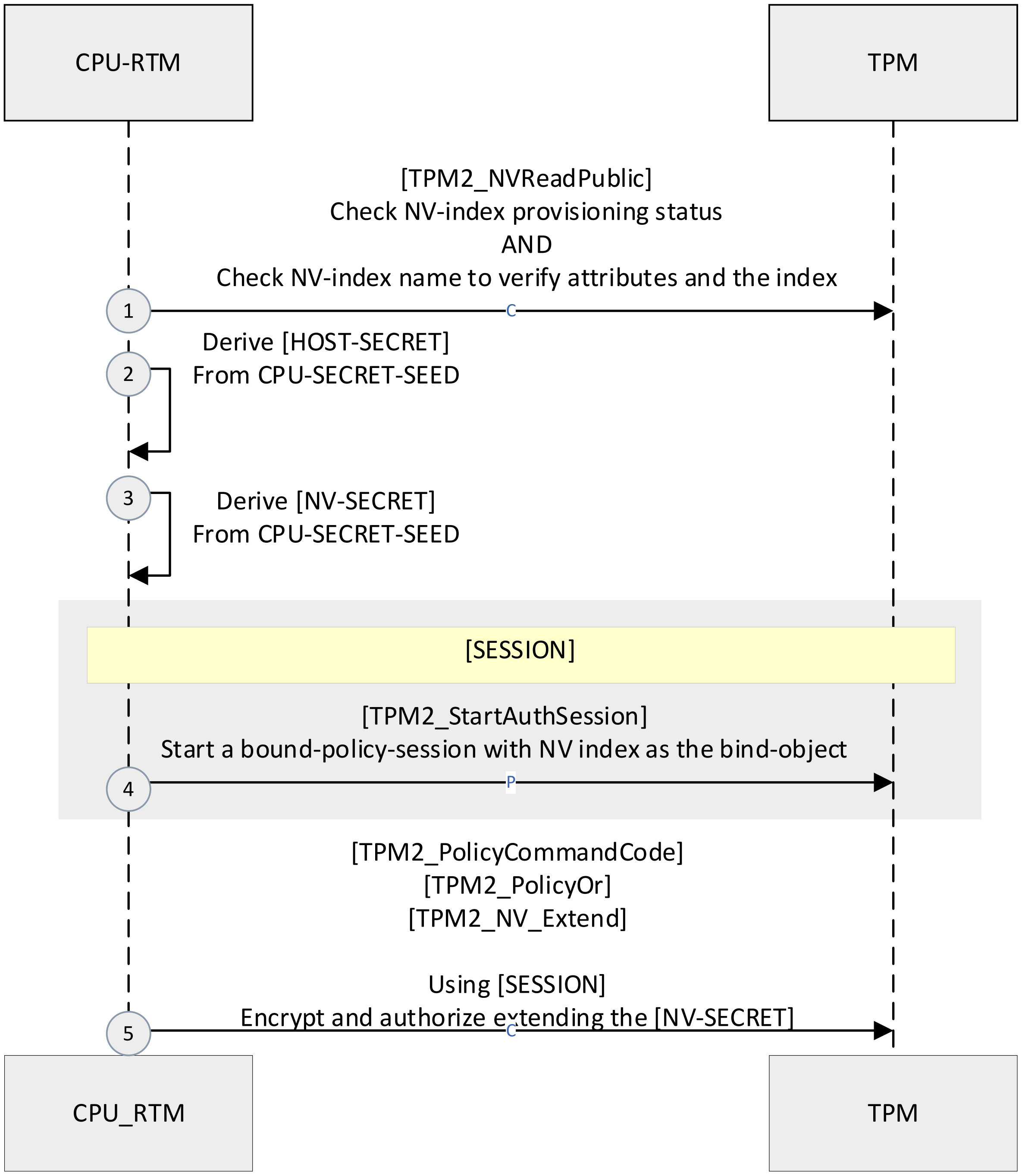

Figure 5 Fulfilling policy for the Seal/UnSeal process

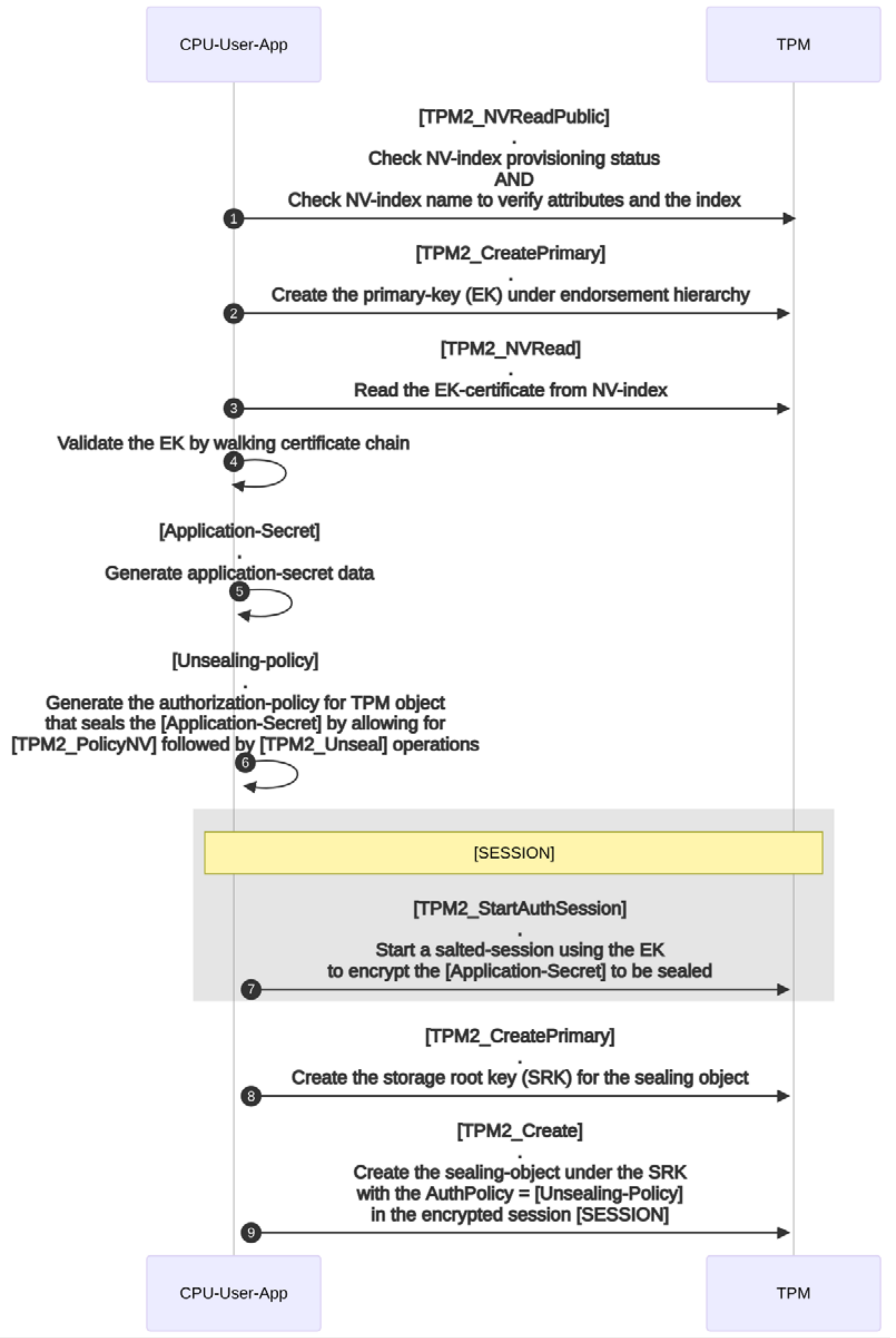
CPU-User-App
TPM
[TPM2_NVReadPublic]
.
Check NV-index provisioning status
AND
Check NV-index name to verify attributes and the index
1
[TPM2_CreatePrimary]
.
Create the primary-key (EK) under endorsement hierarchy
2
[TPM2_NVRead]
.
Read the EK-certificate from NV-index
3
Validate the EK by walking certificate chain
4
[Application-Secret]
.
Generate application-secret data
5
[Unsealing-policy]
.
Generate the authorization-policy for TPM object
that seals the [Application-Secret] by allowing for
[TPM2_PolicyNV] followed by [TPM2_Unseal] operations
6
[SESSION]
[TPM2_StartAuthSession]
.
Start a salted-session using the EK
to encrypt the [Application-Secret] to be sealed
7
[TPM2_CreatePrimary]
.
Create the storage root key (SRK) for the sealing object
8
[TPM2_Create]
.
Create the sealing-object under the SRK
with the AuthPolicy = [Unsealing-Policy]
in the encrypted session [SESSION]
9
CPU-User-App
TPM

Figure 6 Sealing the Secret

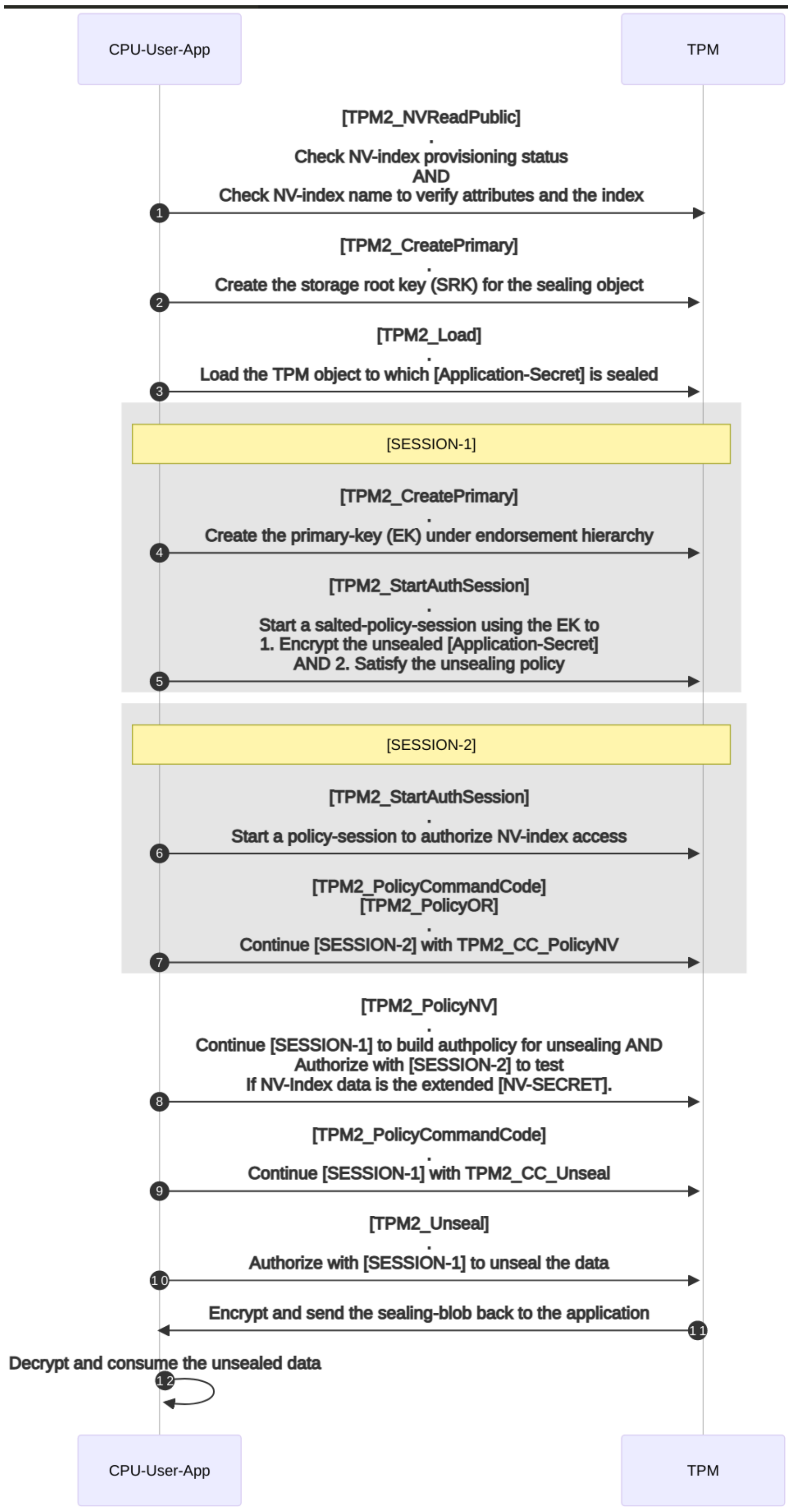
CPU-User-App
TPM
[TPM2_NVReadPublic]
.
Check NV-index provisioning status
AND
Check NV-index name to verify attributes and the index
1
[TPM2_CreatePrimary]
.
Create the storage root key (SRK) for the sealing object
2
[TPM2_Load]
.
Load the TPM object to which [Application-Secret] is sealed
3
[SESSION-1]
[TPM2_CreatePrimary]
.
Create the primary-key (EK) under endorsement hierarchy
4
[TPM2_StartAuthSession]
.
Start a salted-policy-session using the EK to
1. Encrypt the unsealed [Application-Secret]
AND 2. Satisfy the unsealing policy
5
[SESSION-2]
[TPM2_StartAuthSession]
.
Start a policy-session to authorize NV-index access
6
[TPM2_PolicyCommandCode]
[TPM2_PolicyOR]
.
Continue [SESSION-2] with TPM2_CC_PolicyNV
7
[TPM2_PolicyNV]
.
Continue [SESSION-1] to build authpolicy for unsealing AND
Authorize with [SESSION-2] to test
If NV-Index data is the extended [NV-SECRET].
8
[TPM2_PolicyCommandCode]
.
Continue [SESSION-1] with TPM2_CC_Unseal
9
[TPM2_Unseal]
.
Authorize with [SESSION-1] to unseal the data
10
Encrypt and send the sealing-blob back to the application
11
Decrypt and consume the unsealed data
12
CPU-User-App
TPM

Figure 7 Unsealing the secret

3.10.4 Implementation using tpm2-software

```
#!/bin/bash
# This sample requires the TSS and tools available at https://github.com/tpm2-software

set -E

tpm-secret-protection-demo-help() {
    echo "
    #
    # Flows:
    #
    # RTM checks for the NVIndex and its properties and triggers either:
    #
    # (1) cpu_secret_provisioning : Creates NVIndex with required auths & attributes
    #              OR
    # (2) runtime_provisioning    : Extends NVIndex with data only known to the CPU
    #
    # Applications can now reference the NVIndex in a PolicyNV and perform either:
    #
    # (1) seal_data               : Create a sealing object and seal data
    #
    # (2) unseal_data             : Load the sealing object and unseal data
    #
    # NOTE: Source this script to avail all the functions
    "
}

#
# Globals
#
# The NV index should ideally be created as a platform hierarchy object
#
NVIndex=0x1500018

# This is one of the two secrets derived from a seed value ideally accessible
# only to the CPU. This serves as the NV index auth that can be used as bind obj
HOST_SECRET="host-secret"

# This is the second secret derived from a seed value ideally accessible
# only to the CPU. When extended to the NV index, a hash of this value can be
# used in PolicyNV to authorize unsealing of application secret. Only a CPU with
# access to the HOST_SECRET will be able to extend the required value.
NV_SECRET="nv-secret"
```

```
# The application secret to be sealed to the TPM. So long as the CPU extends the
# required NV_SECRET value to the TPM, the unsealing operation is allowed.
SEALBLOB="app-secret"

cleanup() {

    tpm2_clear -Q
    tpm2_nvreadpublic | grep -q $NVIndex
    if [ $? == 0 ]; then
        tpm2_nvundefine -Q -C p $NVIndex
    fi
    rm -f ek.ctx salted_session.ctx policycc_nv_session.ctx A.policy B.policy \
    C.policy nvaccess_policy_generation_session.ctx nvaccess.policy \
    bounded_policy_session.ctx unseal_policy_generate_session.ctx \
    unseal.policy oprim.ctx seal_obj.ctx nvread_session.ctx
}

trap cleanup EXIT

#
# Salted session for encrypting sensitive information when:
# 1. Creating the NV index
# 2. Creating the sealing object
# 3. Unsealing the application secret
#
setup_salted_param_encrypt_session_with_ek() {

    tpm2_createek -Q --key-algorithm rsa --ek-context ek.ctx

    tpm2_startauthsession -Q  --session salted_session.ctx $1 \
    --tpmkey-context ek.ctx
    rm -f ek.ctx

    tpm2_sessionconfig -Q  salted_session.ctx --enable-decrypt
}

#
# Starting the PolicyNV policy session required when:
# 1. Generating the policy for the sealing object at creation
# 2. Unsealing the application secret
#
nvaccess_policycc_policynv() {

    tpm2_startauthsession -Q --session policycc_nv_session.ctx --policy-session
    tpm2_policycommandcode -Q --session policycc_nv_session.ctx TPM2_CC_PolicyNV

    tpm2_policyor -Q --session policycc_nv_session.ctx \
    --policy-list sha256:A.policy,B.policy,C.policy
}

#
# Generate the policy paths for accessing read/ write operations on NV index
# A.policy ==> PolicyCommandCode = TPM2_CC_NV_Read
# B.policy ==> PolicyCommandCode = TPM2_CC_NV_Extend
# C.policy ==> PolicyCommandCode = TPM2_CC_PolicyNV
# Access-Policy = A||B||C
#
generate_nv_access_policy() {

    tpm2_startauthsession -Q --session nvaccess_policy_generation_session.ctx

    tpm2_policycommandcode -Q TPM2_CC_NV_Read --policy A.policy \
    --session nvaccess_policy_generation_session.ctx

    tpm2_flushcontext -Q nvaccess_policy_generation_session.ctx

    tpm2_startauthsession -Q --session nvaccess_policy_generation_session.ctx

    tpm2_policycommandcode -Q  TPM2_CC_NV_Extend --policy B.policy \
    --session nvaccess_policy_generation_session.ctx

    tpm2_flushcontext -Q nvaccess_policy_generation_session.ctx

    tpm2_startauthsession -Q --session nvaccess_policy_generation_session.ctx

    tpm2_policycommandcode -Q TPM2_CC_PolicyNV --policy C.policy \
    --session nvaccess_policy_generation_session.ctx

    tpm2_flushcontext -Q nvaccess_policy_generation_session.ctx

    tpm2_startauthsession -Q --session nvaccess_policy_generation_session.ctx

    tpm2_policyor -Q --session nvaccess_policy_generation_session.ctx \
    --policy-list sha256:A.policy,B.policy,C.policy --policy nvaccess.policy

    tpm2_flushcontext -Q nvaccess_policy_generation_session.ctx

    rm -f nvaccess_policy_generation_session.ctx
```

```
}

#
# This provisioning step is done once under RTM control
#
cpu_secret_provisioning() {
    generate_nv_access_policy

    setup_salted_param_encrypt_session_with_ek --hmac-session

    tpm2_nvdefine -Q  --session salted_session.ctx -C p -p $HOST_SECRET $NVIndex \
    -a
"orderly|clear_stclear|platformcreate|no_da|nt=extend|policyread|policywrite|authread|authwrite"
\
    --policy nvaccess.policy

    tpm2_flushcontext -Q salted_session.ctx
    rm -f salted_session.ctx nvaccess.policy
}

#
# This step is done once at every TPM restart under RTM control
# Satisfy policy to be able to extend the NV index in bound policy session
# Note: Auth specified in the bound session generation is used to
#       calculate the sessionvalue by ESAPI. The auth is not exposed on
#       TPM interface.
#
runtime_provisioning() {
    tpm2_startauthsession -Q  --session bounded_policy_session.ctx \
    --policy-session --bind-context $NVIndex --bind-auth $HOST_SECRET

    tpm2_sessionconfig -Q bounded_policy_session.ctx \
    --enable-decrypt --enable-encrypt

    tpm2_policycommandcode -Q --session bounded_policy_session.ctx \
    TPM2_CC_NV_Extend

    tpm2_policyor -Q --session bounded_policy_session.ctx \
    --policy-list sha256:A.policy,B.policy,C.policy

    echo -n $NV_SECRET|tpm2_nvextend -Q -C $NVIndex -i- $NVIndex \
    -P session:bounded_policy_session.ctx

    tpm2_flushcontext -Q bounded_policy_session.ctx
    rm -f bounded_policy_session.ctx
```

```
}

#
# Satisfy policy to be able to read the NV Index
#
nvread_session_setup() {
    tpm2_startauthsession -Q --session nvread_session.ctx --policy-session

    tpm2_policycommandcode -Q --session nvread_session.ctx TPM2_CC_NV_Read

    tpm2_policyor -Q --session nvread_session.ctx \
    --policy-list sha256:A.policy,B.policy,C.policy
}

#
# Sealing-object-policy:
# PolicyCommandCode == (TPM2_CC_PolicyNV && TPM2_CC_Unseal)
#
seal_data() {

    nvaccess_policycc_policynv

    tpm2_startauthsession -Q --session unseal_policy_generate_session.ctx

    nvread_session_setup

    tpm2_nvread $NVIndex -P session:nvread_session.ctx | tpm2_policynv -Q -i- \
    $NVIndex eq --session unseal_policy_generate_session.ctx \
    -P session:policycc_nv_session.ctx --policy unseal.policy

    tpm2_flushcontext -Q policycc_nv_session.ctx
    tpm2_flushcontext -Q nvread_session.ctx
    rm -f policycc_nv_session.ctx
    rm -f nvread_session.ctx

    tpm2_policycommandcode -Q --session unseal_policy_generate_session.ctx \
    --policy unseal.policy TPM2_CC_Unseal

    tpm2_flushcontext -Q unseal_policy_generate_session.ctx
    rm -f unseal_policy_generate_session.ctx

    setup_salted_param_encrypt_session_with_ek --hmac-session

    tpm2_createprimary -Q  -C o -c oprim.ctx
    echo -n $SEALBLOB | tpm2_create -Q  -C oprim.ctx --policy unseal.policy \
    -u seal_obj.pub -r seal_obj.priv --session salted_session.ctx -i-
    rm -f oprim.ctx
    rm -f unseal.policy

    tpm2_flushcontext -Q salted_session.ctx
    rm -f salted_session.ctx
}

load_sealing_object() {
    tpm2_createprimary -Q  -C o -c oprim.ctx

    tpm2_load -Q  -C oprim.ctx -c seal_obj.ctx -u seal_obj.pub -r seal_obj.priv
    rm -f oprim.ctx
}

#
# 1. Read NV index <non secret data>
# 2. Satisfy PolicyNV
# 3. Unseal
#
unseal_data() {

    nvaccess_policycc_policynv

    setup_salted_param_encrypt_session_with_ek --policy-session

    nvread_session_setup

    tpm2_nvread -C $NVIndex -P session:nvread_session.ctx $NVIndex | \
    tpm2_policynv -Q -i- $NVIndex eq --session salted_session.ctx \
    -P session:policycc_nv_session.ctx

    tpm2_flushcontext -Q policycc_nv_session.ctx
    tpm2_flushcontext -Q nvread_session.ctx
    rm -f policycc_nv_session.ctx
    rm -f nvread_session.ctx

    tpm2_policycommandcode -Q --session salted_session.ctx TPM2_CC_Unseal

    load_sealing_object

    UNSEALBLOB=$(tpm2_unseal -Q -c seal_obj.ctx -p session:salted_session.ctx)

    echo "UNSEALBLOB=$UNSEALBLOB"
    tpm2_flushcontext -Q salted_session.ctx
    rm -f seal_obj.ctx
}

tpm-secret-protection-demo-help
```