IBM TPM 2.0 (and 1.2) TSS API

Ken Goldman

IBM Research

kgoldman@us.ibm.com

August 15, 2024

[1. Introduction 5](#_Toc35934217)

[2. Features 6](#_Toc35934218)

[3. API 7](#_Toc35934219)

[3.1. TSS\_Execute() 7](#_Toc35934220)

[3.2. TSS\_Create() 7](#_Toc35934221)

[3.3. TSS\_Delete() 8](#_Toc35934222)

[3.4. Optional Customization 9](#_Toc35934223)

[3.4.1. Property Example 9](#_Toc35934224)

[3.4.2. Properties 9](#_Toc35934225)

[3.4.3. TSS\_SetProperty() 12](#_Toc35934226)

[3.5. Extra Parameter 13](#_Toc35934227)

[3.6. Other APIs and Headers 13](#_Toc35934228)

[4. Application Notes 15](#_Toc35934229)

[4.1. TPM Simulator 15](#_Toc35934230)

[4.2. Parameter Encryption 15](#_Toc35934231)

[4.3. Session Salt 15](#_Toc35934232)

[4.4. Session Bind 15](#_Toc35934233)

[4.5. NV 16](#_Toc35934234)

[4.5.1. Rationale 16](#_Toc35934235)

[4.5.2. NV Pre-provisioning 17](#_Toc35934236)

[4.6. TPM2\_LoadExternal 17](#_Toc35934237)

[4.7. Connecting to Resource Managers 17](#_Toc35934238)

[4.8. Endorsement Key (EK) Certificates 18](#_Toc35934239)

[4.8.1. Nuvoton 19](#_Toc35934240)

[4.8.2. St Micro 19](#_Toc35934241)

[4.8.3. Infineon 20](#_Toc35934242)

[4.8.4. NationZ 20](#_Toc35934243)

[4.8.5. Intel 20](#_Toc35934244)

[4.8.5.1. Intel EK Certificate Download 20](#_Toc35934245)

[4.9. Command Line Utilities 21](#_Toc35934246)

[4.10. TSS for TPM 1.2 22](#_Toc35934247)

[5. Examples 23](#_Toc35934248)

[5.1. signapp.c 23](#_Toc35934249)

[5.2. writeapp.c 23](#_Toc35934250)

[6. Utility tools 25](#_Toc35934251)

[6.1. Debugging Aids 25](#_Toc35934252)

[6.1.1. reponsecode 25](#_Toc35934253)

[6.1.2. printattr 25](#_Toc35934254)

[6.1.3. timepacket 25](#_Toc35934255)

[6.2. Policy Aids 25](#_Toc35934256)

[6.2.1. policymaker 26](#_Toc35934257)

[6.2.2. policymakerpcr 26](#_Toc35934258)

[6.2.3. publicname 26](#_Toc35934259)

[6.3. Key Manipulation 26](#_Toc35934260)

[6.3.1. createek 26](#_Toc35934261)

[6.3.2. createekcert 27](#_Toc35934262)

[6.3.3. tpm2pem 27](#_Toc35934263)

[6.4. Event Logs 27](#_Toc35934264)

[6.4.1. eventextend 27](#_Toc35934265)

[6.4.2. imaextend 27](#_Toc35934266)

[7. Build 28](#_Toc35934267)

[7.1. Build Options 28](#_Toc35934268)

[7.1.1. TPM\_TPM20 and TPM\_TPM12 28](#_Toc35934269)

[7.1.2. TPM\_POSIX or TPM\_WINDOWS 28](#_Toc35934270)

[7.1.3. TPM\_WINDOWS\_TBSI 28](#_Toc35934271)

[7.1.4. TPM\_TSS\_NOFILE 28](#_Toc35934278)

[7.1.5. TPM\_TSS\_NOCRYPTO 29](#_Toc35934279)

[7.1.6. TPM\_TSS\_NO\_PRINT 29](#_Toc35934280)

[7.1.7. TPM\_TSS\_NOECC 29](#_Toc35934281)

[7.1.8. TPM\_TSS\_NORSA 29](#_Toc35934282)

[7.1.9. TPM\_TSS\_NOENV 29](#_Toc35934283)

[7.1.10. TPM\_NOSOCKET 29](#_Toc35934286)

[7.1.11. TPM\_TSS\_NOCMDCHECK 30](#_Toc35934287)

[7.1.12. TPM\_TSS\_NODEPRECATED 30](#_Toc35934288)

[7.1.13. TPM\_TSS\_NUVOTON 30](#_Toc35934289)

[7.2. Directories 30](#_Toc35934290)

[7.3. Linux 31](#_Toc35934291)

[7.4. Windows 31](#_Toc35934292)

[7.4.1. Windows gcc 33](#_Toc35934293)

[7.4.2. Windows Visual Studio 33](#_Toc35934300)

[7.4.3. Windows Tbsi 34](#_Toc35934301)

[7.5. Mac 34](#_Toc35934302)

[7.6. AIX 35](#_Toc35934303)

[8. Fedora 36](#_Toc35934304)

[8.1. Local Install 36](#_Toc35934305)

[8.2. Alternative Local Install 36](#_Toc35934306)

[8.3. Repository Install 37](#_Toc35934307)

[8.4. Install Test 37](#_Toc35934308)

[8.5. Source rpms 37](#_Toc35934309)

[9. Status 39](#_Toc35934310)

[9.1. Utilities 39](#_Toc35934311)

[9.2. Bugs 39](#_Toc35934312)

[9.3. Untested 39](#_Toc35934313)

[10. Threading 40](#_Toc35934314)

[11. Troubleshooting 41](#_Toc35934315)

[11.1. Environment Variables 41](#_Toc35934316)

[11.2. Command line utilities fail on Windows 10 41](#_Toc35934317)

[11.3. OpenSSL Linking on Windows 41](#_Toc35934318)

[11.4. Loaded objects (keys) disappear 41](#_Toc35934319)

# Introduction

The IBM TSS is designed for:

* ease of understanding
* ease of use
* ease of implementation
* maximum code reuse

The package is upstreamed to

* Fedora
* Debian
* RHEL 7.4 and up
* SLES 12 SP3 and up
* Ubuntu 17.10 and up

# Features

The TSS handles the following, completely hidden from the caller:

* HMAC, password and policy sessions
* Session and HMAC key calculations, including bind and salt sessions
* HMAC generation and verification (including cpHash and rpHash)
* Parameter encryption and decryption, XOR and AES
* Nonces and nonce rolling
* Session continue flag
* TPM 2.0 "Name" and bind session tracking
* Different session hash algorithms
* Marshaling, unmarshaling, and communication with the TPM

and almost hidden from the caller:

* bind password

# API

The API consists of the following calls:

## TSS\_Execute()

#include <ibmtss/tss.h>

TPM\_RC TSS\_Execute(TSS\_CONTEXT \*tssContext,

RESPONSE\_PARAMETERS \*out,

COMMAND\_PARAMETERS \*in,

EXTRA\_PARAMETERS \*extra,

TPM\_CC commandCode,

...);

This is the primary TSS function.

tssContext: Opaque object

out: The standard TPM2 Part 3 response parameter

in: The standard TPM2 Part 3 command parameter

extra: Some commands (only two so far) require extra parameter s.

commandCode: The standard TPM2 Part 2 command code.

. . . : A list of session 3-tuples , of the form

TPMI\_SH\_AUTH\_SESSION sessionHandle,

const char \*password,

unsigned int sessionAttributes

The list is terminated with (TPM\_RH\_NULL, NULL, 0)

## TSS\_Create()

#include <ibmtss/tss.h>

TPM\_RC TSS\_Create(TSS\_CONTEXT \*\*tssContext);

This creates the TSS\_CONTEXT used in the TSS\_Execute() function. It is initialized with the default configuration, which can be then changed using 3.4.3 TSS\_SetProperty().

Returns an error if the context cannot be allocated, or if the properties cannot be initialized, typically due to an invalid environment variable.

See 3.3 TSS\_Delete().

It does not immediately open a connection, so that the connection properties can be changed from the default first.

## TSS\_Delete()

#include <ibmtss/tss.h>

TPM\_RC TSS\_Delete(TSS\_CONTEXT \*tssContext);

The deletes the opaque context created using 3.2 TSS\_Create().

It closes an open connection.

Returns an error if the connection close fails.

## Optional Customization

The TSS is designed to work by default with no configuration.

The current default connects to the Microsoft format socket simulation. This will eventually change to connect to the resource manager.

There are three ways to customize the configuration:

1. At compile time, with a compiler flag
2. At program start, using an environment variable
3. During run time, using the 3.4.3 TSS\_SetProperty() function.

The environment variables and TSS\_SetProperty property use the same names. The makefile flag uses the name with \_DEFAULT appended.

The environment variable overrides the compiler flag, and the TSS\_SetProperty() function overrides both the compiler flag and the environment variable.

### Property Example

To change the default TPM interface to the TPM device driver:

* With a makefile:

-DTPM\_INTERFACE\_TYPE\_DEFAULT="\"dev\""

* With an environment variable:

> setenv TPM\_INTERFACE\_TYPE dev

* With the TSS\_SetProperty() function:

rc = TSS\_SetProperty(tssContext, TPM\_INTERFACE\_TYPE, "dev");

### Properties

(Remember that the makefile compiler flag requires \_DEFAULT to be added, and that the quotes must be escaped.).

The property and legal values are:

#### TPM\_TRACE\_LEVEL

default - 0

0 - no tracing

1 - trace errors

2 - trace errors and execution flow

#### TPM\_DATA\_DIR

default - current directory

set the directory where the TSS can store persistent data

#### TPM\_INTERFACE\_TYPE

default - socsim

socsim - the socket simulator

see

TPM\_SERVER\_NAME

TPM\_SERVER\_TYPE

TPM\_COMMAND\_PORT

TPM\_PLATFORM\_PORT

dev - TPM device driver

see

TPM\_DEVICE

#### TPM\_SERVER\_NAME

default - localhost

set the socket server name (full host name or dotted decimal)

#### TPM\_SERVER\_TYPE

Used with TPM\_INTERFACE\_TYPE = socsim

default - mssim

mssim - send packets in the Microsoft simulator format (header and footer)

raw - send packets in the raw TPM 2.0 specification Part 3 format

rawsingle - same as raw but opens and closes the connection for each command

(useful with the IBM SW TPM 1.2 simulator)

#### TPM\_COMMAND\_PORT

default - 2321

set the socket port for TPM commands

#### TPM\_PLATFORM\_PORT

default - 2322

set the socket port for TPM simulator platform commands

#### TPM\_DEVICE

Unix/Linux default - /dev/tpm0 (single user)

Windows default - Windows 7,8,10 Tbsi

For Unix, sets the TPM device name

/dev/tpmrm0 is the multi-user kernel resource manager

Once the kernel resource manager is upstreamed, this

may become the default.

For Windows, not currently used, only Tbsi supported

#### TPM\_ENCRYPT\_SESSIONS

default 1

1 - Session state is saved encrypted

0 - Session state is saved in plaintext

Since session state can potentially hold secrets, it should normally be encrypted. When the process terminates, the ephemeral encryption key is lost.

See 4.9 Command Line Utilities for the special case of using the command line utilities. That section is not applicable when using the TSS library in programs.

#### TPM\_TRANSMIT\_LOCALITY

default 0

set the locality for transmitting a command.

This property only affects the TPM\_INTERFACE\_TYPET socsim and TPM\_SERVER\_TYPE mssim. That is, the SW TPM packet format permits a locality specifier. HW TPM locality requires hardware support and cannot typically be set by an OS level application.

The main use case for this property is to send TPM2\_Startup at a different locality to simulate the UEFI event EV\_NO\_ACTION, which sets PCR 0 to the startup locality.

### TSS\_SetProperty()

#include <ibmtss/tss.h>

TPM\_RC TSS\_SetProperty(TSS\_CONTEXT \*tssContext,

int property,

const char \*value);

The TSS\_SetProperty() function overrides the defaults and environment variables programmatically.

If the property is related to the connection, an open connection is closed before the property is processed.

NOTE: The close occurs even if the new value is the same as the old value. This can be used to close a connection without deleting the context.

Question: Is it good to mandate this behavior? It offers functionality and makes the implementation easier, but perhaps it's too clever?

NOTE: The value parameter is always a string. For simplicity, the 'value' pointer is stored. The input should be a constant string.

NOTE: For the property TPM\_TRACE\_LEVEL, tssContext is ignored. The trace level is per process, not per context.

## Extra Parameter

The extra parameter is a catch-all for any parameters that TSS\_Execute() requires beyond the normal TPM command and response parameters.

TPM2\_StartAuthSession needs the bind password so that it can calculate the session key.

## Other APIs and Headers

Headers are in the …/utils/ibmtss directory.

#include <ibmtss/tss.h>

The utility / demo applications cheat a bit, in that they call into TSS utility functions. These are less likely to be stable than the official API above.

* tss.h: The official API
* tsserror.h: Included by tss.h for convenience. Error codes may be added.
* tssmarshal.h: Marshal structures to arrays. These are likely to be stable. They are similar to the TPM side functions but return errors.
* Unmarshal\_fp.h: Unmarshal arrays to structures.

The functions without the TSS\_ prefix are deprecated. They were borrowed from the TPM side but used a signed value for the size. The functions prefixed TSS\_ use an unsigned value for the size.

* tssresponsecode.h: Response code to text. Useful for debugging. The API should be stable, but the actual output may change.
* tssprint.h: Functions to print structures. Useful for debugging. The API should be stable. Functions may be added, and the output is likely to change.
* tssprintcmd.h: Functions to print command parameters. Used in TSS verbose tracing. The API should be stable, but the actual output may change.
* tssutils.h: Demo helper functions. These are useful for rapid prototyping but are not recommended for production code.
* tssfile.h: Demo helper functions. These are useful for rapid prototyping but are not recommended for production code.
* tsscrypto.h: Sample crypto code. These are useful for rapid prototyping but are not recommended for production code.
* Any of the lower layer TSS functions are for TSS internal use. They should not be called.

# Application Notes

Several areas have non-obvious usage. They are described here.

## TPM Simulator

A typical cause of a hang when sending the first command to the TPM simulator is that it has not received a simulated "powered up." The IBM TPM simulator does not require this, but others may. Send this command:

> powerup

The platform firmware initializes a hardware TPM. The TPM simulator requires this command:

> startup

## Parameter Encryption

The caller does NOT perform parameter encryption. Simply set the session attribute to either or both of TPMA\_SESSION\_ENCRYPT or TPMA\_SESSION\_DECRYPT.

## Session Salt

To salt, the caller should set tpmKey (the handle of a loaded decrypt key) in TPM2\_StartAuthSession. The key must be an RSA 2048-bit key or ECC NIST P256 key with sign clear and decrypt set.

The caller must supply the extra parameter as a StartAuthSession\_Extra structure.

The caller does NOT supply the HMAC salt. The encryptedSalt parameter is ignored, as the TSS generates the salt.

## Session Bind

To bind, the caller should set bind (the bind entity handle) in TPM2\_StartAuthSession. The caller must supply the extra parameter as a StartAuthSession\_Extra structure and set the bindPassword member to the bind handle password.

## NV

For applications that do not share an NV index and don't use global locks or transient locks or written status that change after a reboot, the following details are unnecessary. Just issue the TPM2\_NV\_DefineSpace and everything works, including HMAC sessions.

For applications that do not satisfy the above criteria, the application must issue TPM2\_NV\_ReadPublic and validate that the public area is as expected, including the locks.

This will not typically occur in real applications. The transient locks and written status permit low level firmware to perform pre-OS operations. By the time post-OS applications are running, the locks and written status should be restored.

### Rationale

Why? For authorization, the TSS includes the Name (a hash of the public NV metadata) in the HMAC calculation. This ensures that the NV index has not been replaced with a different version with untrusted metadata.

Normally, the TSS tracks the Name, even when the application changes the written or lock attributes. However, if the attributes (and therefore the Name) change outside the application, the TSS is unable to track the change. In those cases, the application must read the public data and validate it.

Why doesn't the TSS automatically issue the TPM2\_NV\_ReadPublic? If it did that, it would encourage the application developer to blindly trust the index. The application is expected to examine the TPM2\_NV\_ReadPublic return (e.g., the policy, the attributes) and decide whether the index is trusted. The TSS cannot enforce this, but it does at least encourage it.

The index Name can change as the metadata changes. These changes include the "written" bit and the read and write locks. The TSS automatically tracks the changes performed on a single index by the application. It does not track:

1. Changes "out of band", by a different application, including an attacker.

This will surface as an HMAC failure. The application should reissue TPM2\_NV\_ReadPublic and decide if the index is still trusted.

1. Changes to transient lock status due to a reboot.

If the application is aware of the reboot, it can reissue TPM2\_NV\_ReadPublic and re-evaluate the return. Otherwise, it can wait for the potential HMAC failure and handle it as above.

1. Locks due to a global lock, because it's hard and because global lock is expected to be used only at provisioning, if ever.

This case will probably never occur in practice. If it does, handle the HMAC failure as above.

### NV Pre-provisioning

This is the case where some other application has used TPM2\_NV\_DefineSpace to pre-provision an index. As an alternative to the application evaluating the TPM2\_NV\_ReadPublic response, the NV metadata and Name can be pre-provisioned when the application is installed. Two files are required:

[01nnnnnn] is the hex value of the NV index.

1. h[01nnnnnn].bin - The Name, a binary hash of the public data
2. nvp[01nnnnnn].bin - The marshaled TPMS\_NV\_Public

## TPM2\_LoadExternal

This command is unique, in that it has an optional parameter, TPM2B\_SENSITIVE *inPrivate*. The caller should use the size as a flag: 0 for not present, and non-zero for present.

Rationale:

The TPM uses the *inPrivate.size* zero to indicate that the parameter is not present and uses the correct marshaled size to indicate that the parameter is present. This TSS uses that design pattern, but, as with other TPM2B's that wrap structures, it does not require the caller to marshal the structure and determine the correct size.

## Connecting to Resource Managers

An issue arises when using the TSS utilities (not the TSS itself) on a platform with a resource manager. Windows 10 has a resource manager called TBS and Linux as of 4.12 uses /dev/tpmrm0.

A resource manager flushes all resources (objects like keys, and sessions) when a connection closes. Since the utilities are standalone processes, the connection closes after each invocation. Thus, for example, a utility can load a key, but, when the "load" command terminates, the resource manager will flush the key.

This is not an issue for a complete application using the TSS because the TSS keeps the connection open through multiple TPM commands. For prototyping using the utilities, the Linux solution is to bypass the resource manager using /dev/tpm0. For Windows 10, use a proxy to simulate this persistent connection behavior.

The "tpmproxy" program connects as a socket server on one side and a TPM device driver on the other. Once the proxy starts, the Windows resource manager sees one persistent connection, as desired. The utilities use the socket interface to the proxy.

Both the TSS and the proxy support both a raw packet format and the Microsoft simulator wrapped format.

The default, using the wrapped format, can also be specified on the TSS side with

> TPM\_INTERFACE\_TYPE=socsim

> TPM\_SERVER\_TYPE=mssim

and the tpmproxy with -mssim.

The raw format can be specified with

> TPM\_INTERFACE\_TYPE=socsim

> TPM\_SERVER\_TYPE=raw

and the tpmproxy with -raw.

## Endorsement Key (EK) Certificates

The TSS includes several TPM vendor EK root certificates for convenience.

There is no reason for a user to trust these certificates. Obtain production certificates directly from the TPM vendor.

These URLs are provided for convenience. Observe that some URLs are http. I encourage all readers to ask the TPM vendors to offer these certificates over a secured web page, since they form the root of trust for TPM authenticity.

Utilities such as "createek" that take a -root argument require a list of EK root certificates in a file. The utilities include a sample file …/utils/certificates/rootcerts.txt. The file MUST be edited, since the file names must have a complete path to your install directory.

The file must have a Unix newline (only LF, 0x0a), not a DOS newline, even on Windows platforms.

Certificates must be in PEM format. To convert from DER format (.cer, .crt) to PEM using openssl:

> openssl x509 -in cert.cer -inform DER -out cert.pem

### Nuvoton

<https://www.nuvoton.com/security/NTC-TPM-EK-Cert/Nuvoton%20TPM%20Root%20CA%202110.cer>

<https://www.nuvoton.com/security/NTC-TPM-EK-Cert/Nuvoton%20TPM%20Root%20CA%201110.cer>

<https://www.nuvoton.com/security/NTC-TPM-EK-Cert/Nuvoton%20TPM%20Root%20CA%202111.cer>

https://www.nuvoton.com/security/NTC-TPM-EK-Cert/NuvotonTPMRootCA1210.cer

https://www.nuvoton.com/security/NTC-TPM-EK-Cert/Nuvoton TPM Root CA 2112.cer

https://www.nuvoton.com/security/NTC-TPM-EK-Cert/Nuvoton TPM Root CA 1111.cer

### St Micro

This list was extracted from http://www.st.com/resource/en/data\_brief/stsw-tpmcert1.pdf

GlobalSign Trusted Computing CA

http://secure.globalsign.com/cacert/gstpmroot.crt

ST TPM Root certificate

http://secure.globalsign.com/cacert/stmtpmekroot.crt

ST Intermediate CA 01

http://secure.globalsign.com/cacert/stmtpmekint01.crt

ST Intermediate CA 02

http://secure.globalsign.com/cacert/stmtpmekint02.crt

ST Intermediate CA 03

http://secure.globalsign.com/cacert/stmtpmekint03.crt

ST Intermediate CA 04

http://secure.globalsign.com/cacert/stmtpmekint04.crt

ST Intermediate CA 05

http://secure.globalsign.com/cacert/stmtpmekint05.crt

GlobalSign Trusted Platform Module ECC

Root CA

http://secure.globalsign.com/cacert/tpmeccroot.crt

STM TPM ECC Root CA 01

http://secure.globalsign.com/stmtpmeccroot01.crt

STM TPM ECC Intermediate CA 01

http://secure.globalsign.com/stmtpmeccint01.crt

### Infineon

<https://www.infineon.com/cms/en/product/promopages/optiga_tpm_certificates>

### NationZ

Root

https://pki.nationz.com.cn/EkRootCA/EkRootCA.crt

Intermediate certificates

https://pki.nationz.com.cn/EkMfrCA001/EkMfrCA001.crt

https://pki.nationz.com.cn/EkMfrCA002/EkMfrCA002.crt

https://pki.nationz.com.cn/EkMfrCA003/EkMfrCA003.crt

### Intel

PTT EK Root Certificate

https://upgrades.intel.com/content/CRL/ekcert/EKRootPublicKey.cer

PTT EK Intermediate Certificate

<http://upgrades.intel.com/content/CRL/ekcert/SPTHEPIDPROD_EK_Platform_Public_Key.cer>

### Intel EK Certificate Download

As of June 2017, the Intel PTT did not come provisioned with EK certificates. They must be downloaded using this procedure.

* Read the EK public key
* Construct digest data
  + For RSA, concatenate the public modulus to the default exponent 010001, all in binary.
  + For ECC, this step is currently undocumented.
* Calculate a SHA-256 digest of the digest data
* Base64 encode the digest
* Convert the base64 to URL base64 by changing = to %3D, + to -, and / to \_.
* Prepend <https://ekop.intel.com/ekcertservice/> to form the certificate URL.
* Use a browser to display the certificate (or use wget and edit in a text editor)
* Extract the text between the <certificate> and </certificate> to a text editor.
* Convert the URL base64 to base64 by changing %3D to =, - to +, and \_ to /. Remove all newlines.
* Base64 decode to create the DER certificate.

## Command Line Utilities

See also section 4.7 Connecting to Resource Managers.

As stated in section 3.4.2 Properties, the default is to build the TSS library to encrypt session state with an ephemeral encryption key that is lost when the application exits.

This behavior would preclude using the command line utilities with sessions, since the encryption key would change. There are two facilities to remedy this.

1. For stand-alone debugging, save session state in plaintext. For example, use an environment variable:

> setenv TPM\_ENCRYPT\_SESSIONS 0 (csh, tcsh)

> export TPM\_ENCRYPT\_SESSIONS=0 (bash)

> set TPM\_ENCRYPT\_SESSIONS=0 (windows)

or the equivalent compile time flag

-DTPM\_ENCRYPT\_SESSIONS\_DEFAULT="\"0\""

1. For using the command line utilities securely, either stand-alone or in scripts, a fixed encryption key can be specified.

In the script:

TPM\_SESSION\_ENCKEY=`./getrandom -by 16 -ns`

On the command line:

> setenv TPM\_SESSION\_ENCKEY `./getrandom -by 16 -ns`

## TSS for TPM 1.2

This is new code in 2018. The TSS for TPM 1.2 uses the identical API as in section 3 API, but with TPM 1.2 structures and ordinals.

There is currently support for about 20 TPM 1.2 commands, specifically those commands that are required to implement an attestation client. It also includes support commands to create a SW TPM EK certificate and the activate identity blob, and to extend firmware and IMA event logs into a TPM for testing. More commands can be easily added upon user request.

To build a combined TPM 2.0 and TPM 1.2 TSS:

> make -f makefiletpmc clean all

To build a TPM 2.0 only TSS

> make -f makefiletpm20 clean all

To build a TPM 1.2 only TSS

> make -f makefiletpm12 clean all

The utilities and a basic regression test (reg.sh) are in the ../utils12 directory.

The regression test requires a software TPM 1.2 that is 'straight from the factory' - with no TPM state. The reason is that creating the standard TCG EK certificate requires an unlocked TPM. Once NV is locked, it can never be unlocked. Before running the regression test, remove the state and restart the TPM.

The standard TPM 1.2 is opt-in and is shipped disabled and deactivated. While the IBM software TPM comes with tools to enable and activate the TPM, users may find it less of a nuisance to use 'makefile-en-ac' to build a TPM that is already enabled and activated at its first start.

# Examples

Each standalone utility serves as an example for a single command.

There are also two examples that show how several commands can be chained together to form an application

## signapp.c

The signapp.c source demonstrates how the TSS handles bind and salt sessions. It uses sample code to create an EK. It uses a policy session to authorize the EK. It authorizes a signing key using both HMAC and policy sessions, where the policy can be policy password or policy authvalue. The signing key policy also has an AND term for the command code 'sign'.

* Create an EK for the salt
* Start an authorization policy session, salt with the EK
* Use policysecret to authorize the EK
* Create a signing key under the EK, permit password and policy authorization
* Restart the policy session
* Use policysecret to authorize the EK
* Load the signing key under the EK, using the session
* Create an HMAC session, salt with the EK, bind with the signing key
* Sign a digest, using the HMAC
* Verify the signature
* Restart the policy session
* Use policy command code and policy authvalue to authorize the signing key
* Sign a digest
* Verify the signature
* Flush the policy and HMAC sessions
* Flush the EK primary key
* Flush the signing key

## writeapp.c

The writeapp.c source demonstrates how the TSS handles bind and salted session, and tracks entity Name changes without application coding.

* Create an EK for the salt
* Start a session, salt with EK
* Define an NV index, salted session
* Flush the session
* Start a session, salt with EK, bind to unwritten NV index
* Write NV, changes the Name, bound, salt, encrypt session
* Start a session, salt with EK, bind to written NV index
* Write NV, bound, salt, encrypt session
* Undefine NV index
* Flush EK

# Utility tools

In addition to the command line tools for each TPM command, there are several utilities that are useful for development.

They accept various digest and asymmetric algorithms and other options as applicable. Use -h for details.

## Debugging Aids

### reponsecode

Prints a TPM or TSS hex response code as text.

### printattr

Prints a TPM hex attribute as text.

It supports object, session, startup, and NV attributes.

### timepacket

This is useful for profiling - timing the duration of a single command.

It is occasionally useful to debug a command packet obtained from a source other than the TSS.

It takes a hex ascii input command string so that TSS processing is excluded. The string can be obtained by running the command first with -v and capturing the TSS trace.

This tool is restricted to commands that:

* can be run repeatedly. I.e., it cannot use policy sessions.
* do not include varying data. E.g., it cannot have an HMAC with rolling nonces.

## Policy Aids

These tools provide primitive aids to calculating and debugging policies. The regression test script comments explain how the tools were used to calculate the test policies. The utils/policies directory holds both the 'source' hexascii and binary policies.

### policymaker

This tool accepts a set of hex ascii AND terms, one per line, and calculates the resulting policy. An empty policyRef is represented by a blank line.

The result can be traced and/or output in binary in a format directly usable as a utility input.

-v traces the intermediate terms. In combination with policygetdigest, it can be used to debug a policy term by term.

-ns (no white space) traces the output in a format that can be used as input to a policy OR (which is just a concatenation of AND terms).

-nz calculates the hash without the normal 'extend starting with zeros', useful for calculating an 'aHash' such as a cpHash.

### policymakerpcr

This tool calculates a policypcr AND term in a format suitable for input to policymaker. It accepts a bit mask of selected PCRs and a white list of PCR values, one per line.

### publicname

This tool calculates a TPM Name from a TPM object or NV public structure, or from a PEM or DER format public key.

This is useful for constructing policies at times that the TPM is not available to calculate the Name.

## Key Manipulation

### createek

This tools aggregates several EK manipulation tasks:

* Creates an EK primary key based on TCG standards.
* Prints an EK template or nonce if provisioned.
* Prints an EK certificate. This function is also integrated into nvread.
* Validates an EK certificate against the TPM vendor root certificates.

### createekcert

This tools provisions an EK certificate in TPM NV for testing.

The EK is generated based on TCG standards.

### tpm2pem

This tools converts an existing TPM format public key to PEM format.

This function is also integrated into several tools, such as create, createprimary, createloaded, and readpublic.

## Event Logs

The package has sample functions for parsing pre-OS and post-OS (IMA) event logs. These tools aggregate some of these functions.

### eventextend

This tool parses a pre-OS (BIOS, UEFI) event log. It can:

* Extend the events into a TPM, simulating pre-OS firmware.
* Calculate expected TPM PCR values.
* Calculate a boot aggregate.

### imaextend

This tool parses a post-OS Linux IMA log. It can:

* Extend the events into a TPM, simulating post-OS IMA software.
* Calculate expected TPM PCR values.

To permit scripting a test platform, it can accept beginning and ending events.

# Build

The builds for Linux and Windows create the TSS shared object / dll and about 110 command line programs. The command line programs can be used in a script for rapid prototyping or as sample usage code.

There are currently makefiles for common build options, listed later.

The build files clear TPM\_ENCRYPT\_SESSIONS, which is useful for prototyping and regression testing. This should be removed for production applications.

## Build Options

Several compile time macros permit building a variation of the TSS library. Features may be lost, but the tradeoff may be important in some environments.

**Since the regression test does not function against all builds, some variations are very lightly tested. Please report bugs.**

### TPM\_TPM20 and TPM\_TPM12

Define one or both of these for a TSS that supports TPM 2.0 and/or TPM 1.2.

TPM 1.2 support is incomplete. See section 4.10 TSS for TPM 1.2.

### TPM\_POSIX or TPM\_WINDOWS

Define one of these for a POSIX (Linux, AIX, Raspian, zLinux, etc.) or Windows TSS.

### TPM\_WINDOWS\_TBSI

For Windows, compiles in hardware TPM support.

This was useful to remove hardware TPM support for Windows 7, where the TPM 2.0 install required several hacks. Since Windows 10 supports TPM 2.0, there is little reason not to define this macro.

### TPM\_TSS\_NOFILE

Defining this macro builds a TSS library that does not use files for temporary and persistent state. All state is stored in the TSS context and is lost when the context is deleted.

Drawbacks:

* Scripting, which requires state to persist between processes, does not work.
* Names and public keys of persistent entities do not persist, so the entities must be reread (and revalidated) at each connection.
* Context save and load are not implemented yet.
* There are currently some fixed size arrays for transient object and session state.

### TPM\_TSS\_NOCRYPTO

Defining this macro builds a TSS library that does not depend on a crypto library.

Drawbacks:

* Salted sessions do not work.
* HMAC sessions do not work, including policies that require HMAC.
* Encrypt and decrypt sessions do not work.

### TPM\_TSS\_NO\_PRINT

Defining this macro builds a TSS that does no tracing and compiles out all print functions.

### TPM\_TSS\_NOECC

Defining this macro builds a TSS that does not require crypto library elliptic curve support.

### TPM\_TSS\_NORSA

Defining this macro builds a TSS that does not require crypto library RSA support.

### TPM\_TSS\_NOENV

Defining this macro builds a TSS that does not call the getenv() function. This supports platforms that do not implement environment variables.

The TSS properties still use defaults that can be changed at build time, and it still supports the TSS\_SetProperty() function.

### TPM\_NOSOCKET

Defining this macro builds a TSS that does not include the socket interface. This supports platforms that do not implement sockets.

When this macro is defined, TPM\_INTERFACE\_TYPE defaults to dev, not socsim.

### TPM\_TSS\_NODEV

Defining this macro builds a TSS that does not include the TPM device driver interface.

The main use for this macro is a Windows platform using mingw. mingw does not apparently support the Tbsi interface.

### TPM\_TSS\_NOCMDCHECK

Defining this macro builds a TSS that does not implement input parameter checking. The commands are sent as-is to the TPM, which will presumably do its own check.

This two main use cases of this macro are:

* to reduce the size of the TSS library by about 10%.
* to permit the TSS to send error cases for TPM testing.

The main disadvantage of this macro is that debugging is harder. Rather setting breakpoints in the TSS to debug, the developer must rely on the TPM return codes.

### TPM\_TSS\_NODEPRECATED

Defining this macro removes deprecated TPM functions. This slightly reduces the library size in cases where the application is not using the functions.

### TPM\_TSS\_NODEPRECATEDALGS

Defining this macro removes deprecated crypto algorithms from both the TSS and the command line utilities. If the algorithm is used, the TSS returns an error.

Currently, the only algorithm removed is SHA-1.

To skip those algorithms in the regression test, set the environment variable TPM\_TSS\_NODEPRECATEDALGS to 1.

### TPM\_TSS\_NUVOTON

Defining this macro adds TSS support for the Nuvoton proprietary TPM configuration commands.

## Directories

The TSS ships with these directories:

…/utils TSS for both TPM 1.2 and TPM 2.0.

TPM 2.0 utility / demo applications

…/utils/regtests TSS for TPM 2.0 regression tests

…/utils/policies TSS for TPM 2.0 regression test policies and miscellaneous files

…/utils/certificates TPM for TPM 2.0 and TPM 1.2 vendor EK root certificates

…/utils12 TPM 1.2 utility / demo applications

TSS for TPM 1.2 regression test

…/demo TSS for TPM 2.0 demo web pages

…/tpmutils TSS for TPM 2.0 Visual Studio files

To extract the tarball

> cd …

> tar xvf ibmtssnnn.tar .

## Linux

These are the mainstream TPM 2.0 instructions.

Install OpenSSL 3.x., including the development package.

> cd …/utils

> make -f makefiletpm20

Note: Linux builds must have TPM\_POSIX defined.

After building, run the regression test against a running simulator. See 7.5 Regression Test.

The TPM device driver normally does not permit non-root access. Either

> chmod 777 /dev/tpm0

or run as root or sudo.

## Windows

Read all of this. It’s tricky.

Install OpenSSL 3.x 64-bit. Install Win64 OpenSSL, not the "Light" versions, which I believe do not contain the development files. The usual place to get OpenSSL binaries for Windows is:

<http://slproweb.com/products/Win32OpenSSL.html>

Recent Shining Light installs point to

C:\OpenSSL-Win364

Change this to

C:\Program Files\OpenSSL

If you chose not to install OpenSSL in this location, you must fix the build paths. In other words, use the recommended location.

This directory should be added to the Path environment variable if it's not already there:

c:\Program Files\OpenSSL\bin

Note: Windows builds must have TPM\_WINDOWS defined.

After building, run the regression test against a running simulator. See 7.5 Regression Test.

### Windows gcc

A mingw (Minimalist GNU for Windows) makefile.mak is included.

Issues:

* mingw does not seem compatible with the Windows 10/11 TBSI. Thus, the build does not have hardware TPM support.

For these reasons, the Visual Studio project is recommended.

Contributions to fix this is welcome.

> cd …/utils

> make -f makefile.mak

### Windows Visual Studio

VS solution and project files are supplied. The Visual Studio 2022 Solution is at…/tpmutils/tpmutils.sln.

The default is to connect to the socket simulator using the Microsoft simulator packet format. To change the default from a SW TPM to a HW TPM, add the preprocessor definition:

TPM\_INTERFACE\_TYPE\_DEFAULT="dev"

### Windows Visual Studio Configuration

For other versions of Visual Studio, these build options may work. They have not been tested.

The common property sheets for the TSS and command line utilities are:

Debug: tpmutils/CommonPropoerties64.props

Release: tpmutils/CommonProperties64Release.props

**C/C++ Compiler:**

Additional Include Directories:

c:/program files/openssl/include;../../utils/

Preprocessor Definitions:

TPM\_WINDOWS;TPM\_TPM20;\_CRT\_SECURE\_NO\_WARNINGS;\_WINSOCK\_DEPRECATED\_NO\_WARNINGS;TPM\_ENCRYPT\_SESSIONS\_DEFAULT="0";TPM\_WINDOWS\_TBSI;\_WIN8; TPM\_WINDOWS\_TBSI\_WIN8

**Linker:**

The Shining Light directory structure changes periodically.

In 3.1, all the .lib files were in one directory with different

names. In 3.2, all the .lib files have the same name but are in

different directories.

The dll remains in the bin directory, so the gcc makefile.mak is

unchanged.

This project supports the 3.2 layout in Visual Studio. To change

between them, edit the property sheets.

OpenSSL 3.1

Additional dependencies

Debug libcrypto64mdd.lib

Release libcrypto64md.lib

Additional Library Directories

c:\program files\openssl\lib\vc

OpenSSL 3.2

Additional dependencies

libcrypto.lib

Additional Library Directories

Debug c:\program files\openssl\lib\vc\x64\MDd

Release c:\program files\openssl\lib\vc\x64\MD

## Regression Test

After building for Windows or Linux, run the regression test against a running simulator.

**The regression test does not run against a hardware TPM**, since the platform firmware will have set the platform authorization. There are likely to be other errors due to protected or unsupported TPM features.

**Use the regression test for TSS verification, not as a TPM test tool.**

**Windows**, takes about 15 minutes:

The Windows script assumes that typical command line tools such as touch and diff are installed. A typical download location is

<http://gnuwin32.sourceforge.net/packages.html>

See CoreUtils and DiffUtils.

The regression test script defaults to the executables being in the same directory as the script, …/tpm2/utils. This is correct for the gcc build, but not for the Visual Studio build. To point to those executables, set this environment variable. **Do not omit the trailing slash**.

> set TPM\_EXE\_PATH=..\tpmutils\x64\Debug\

> set TPM\_EXE\_PATH=..\tpmutils\x64\Release\

> reg,bat

**Linux**, takes about 5 minutes:

> ./reg.sh -a

Selective tests can be run. See reg.sh -h for the test numbers.

The regression test can run against a **software** TPM at /dev/tpm0. It will skip the power up sequence. However, it uses the environment variable TPM\_INTERFACE\_TYPE as the determination. If the default TPM\_INTERFACE\_TYPE was changed at compile time, the regression test will try the power up sequence unless the environment variable is also set.

## Mac

This is a contribution. I did not test it.

* install homebrew
* install openssl -> brew install openssl
* install gawk -> brew install gawk
* set PATH of shell (terminal):

PATH=/usr/local/Cellar/openssl/1.0.2m/bin/:$PATH

PATH=/usr/local/Cellar/gawk/4.2.0/bin/:$PATH

Make this permanent by adding to a profile.

* Build using makefile.mac

## AIX

Use gnu make (gmake), not make.

> cd …/utils

> gmake -f makefile.aix

After building, run the regression test against a running Microsoft simulator. -h gives help. Since the TPM simulator does not run on AIX yet, set the TPM\_SERVER\_NAME environment variable.

> reg.sh -a

# Fedora

This section is only relevant to a Fedora rpm install. It is a work in progress and may not be 100% correct yet.

Prerequisite:

# yum install rpm-build

## Local Install

Download the rpms:

TBD

1. Install binaries:

* the libraries - /usr/lib64/libibmtss.so.0.1and the link /usr/lib64/libibmtss.so.0
* the utilities - /usr/bin/tssxxx. Note that the installed utilities are namespaced with the 'tss' prefix.
* the license - /usr/share/doc/ibmtss-nnn/LICENSE

# rpm -ivh ibmtss-nnn-1.el6.x86\_64.rpm

1. Install development headers:

* the headers - /usr/include/ibmtss
* the library - link /usr/lib64/libibmtss.so
* this documentation - /usr/share/doc/ibmtss-devel-nnn/ibmtss.doc

# rpm -ivh ibmtss-devel-nnn-1.el6.x86\_64.rpm

1. Install debug source and support

# rpm -ivh ibmtss-debuginfo-nnn-1.el6.x86\_64.rpm

## Alternative Local Install

Erase an old version as needed:

# yum erase ibmtss-devel-nnn-1.el6.x86\_64

# yum erase ibmtss-nnn-1.el6.x86\_64

# yum erase ibmtss-debuginfo-nnn-1.el6.x86\_64

Install (new method)

# dnf install ./ibmtss-nnn-1.el6.x86\_64.rpm

# dnf install ./ibmtss-devel-nnn-1.el6.x86\_64.rpm

# dnf install ./ibmtss-debuginfo-nnn-1.el6.x86\_64.rpm

Install (old method)

# yum install ./ibmtss-nnn-1.el6.x86\_64.rpm

# yum install ./ibmtss-devel-nnn-1.el6.x86\_64.rpm

# yum install ./ibmtss-debuginfo-nnn-1.el6.x86\_64.rpm

## Repository Install

Once the packages have been upstreamed, use this process.

# dnf install ibmtss

## Install Test

This assumes that the SW TPM has been installed, see this link:

<https://sourceforge.net/projects/ibmtpm20tss/?source=navbar>

It also assumes that the regression test has been installed. See Section 8.5.

In reg.sh, change the utility prefix variable to tss.

PREFIX=tss

Run the regression test:

> cd ~/rpmbuild/BUILD/ibmtss-nnn/utils

> ./reg.sh

## Source rpms

Install source (as non-root user)

> rpm -ivh ibmtss-nnn-1.el6.src.rpm

The src rpm has a tarball and spec file.

# Status

## Utilities

The utilities serve several purposes:

* The utilities are called by a Unix shell or bat script to form the regression test.
* The utilities are sample code on how to use the TSS.
* The utilities can be used in a script for rapid prototyping.
* The regression test scripts are sample code for how to use the utilities and the TPM to perform multi-step tasks.

NOTE: The utility command line arguments are not stable. They change occasionally to improve consistency among utilities or to add features

The utilities currently do not permit all TPM command options. Let me know what needs enhancement.

## Bugs

Please report bugs.

## Untested

These may "just work" but they have not been tested yet.

Users are welcome to suggest ECC tests and prioritize the below list.

* ECC commands - ECDH\_KeyGen, ECDH\_ZGen
* PolicyLocality
* TestParams

# Threading

The TSS is not thread safe.

There are many issues with making a TSS thread safe, because the TPM is inherently single threaded. For example:

* There is only one channel to a TPM. Two threads writing bytes to a socket to a resource manager or simulator, or writing bytes to the device driver, will fail.
* The TPM has session state that has to be coordinated with an application. For example, if a thread begins to calculate an HMAC for a session, and another thread uses the session, the rolling nonces will cause the first thread HMAC to fail.
* Applications have state at a higher level. For example, if a thread begins to use a key and another thread saves the key context and flushes the key, the first thread's application will fail.

I think the best we can do is provide a common "TSS lock semaphore" mechanism, so that threads can coordinate access to the TSS using a common API.

# Troubleshooting

This section includes some frequent issues and solutions.

## Environment Variables

The instructions in 3.4 Optional Customization and elsewhere are often specific to one Unix shell. See your shell documentation for variations. Windows uses yet another syntax.

Settings are local to one process (to one window). When the process exits (when the windows is closed), the setting is lost. To create a persistent setting, use a dotfile (Unix) or a control panel setting (Windows.)

## Command line utilities fail on Windows 10

Windows blocks executables with the strings setup, install, update, and patch in the name. Thus, TPM utilities like sequenceupdate.exe will not run.

One work around is to run the commands shell as administrator. Right click "Command Prompt" and select "Run as administrator".

## OpenSSL Linking on Windows

A failure linking with OpenSSL on Windows is very often caused by multiple versions of OpenSSL installed on the platform. The easiest solution is to run the uninstaller, delete all the OpenSSL directories, and then install just once.

The Windows 10 crypto library has function names that clash with OpenSSL, particularly in the area of X.509 support. Visual Studio includes it by default when using. To remove those headers, define WIN32\_LEAN\_AND\_MEAN. Use the command line utilities as samples.

## Loaded objects (keys) disappear

If an object such as a key is loaded successfully in a script but then seems to disappear, it is likely the interaction with the resource manager. The resource manager detects that the process (one line of the script) exits and then frees all allocated resources.

This occurs with a hardware TPM - always on Windows and with Linux when connecting to /dev/tpmrm0. It will not occur when connecting directly with a software TPM or a hardware TPM at /dev/tpm0, which bypasses the resource manager. It will also not occur once the prototyping script is replaced by an executable that does not close the connection after each TPM command.

The solution is to use a proxy, which keeps the TPM connection alive. See section 4.7 Connecting to Resource Managers.