



Full credit is given to the above companies including the OS that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'tpm2_duplicate.1'

\$ man tpm2_duplicate.1

tpm2_duplicate(1) General Commands Manual tpm2_duplicate(1)

NAME

tpm2_duplicate(1) - Duplicates a loaded object so that it may be used in a different hierarchy.

SYNOPSIS

tpm2_duplicate [OPTIONS]

DESCRIPTION

tpm2_duplicate(1) - This tool duplicates a loaded object so that it may be used in a different hierarchy. The new parent key for the duplicate may be on the same or different TPM or TPM_RH_NULL.

OPTIONS

These options control the key importation process:

? -G, --wrapper-algorithm=ALGORITHM:

The symmetric algorithm to be used for the inner wrapper. Supports:

? aes - AES 128 in CFB mode.

? null - none

? -i, --encryptionkey-in=FILE:

Specifies the filename of the symmetric key (128 bit data) to be used

for the inner wrapper. Valid only when specified symmetric algorithm is not null

? -o, --encryptionkey-out=FILE:

Specifies the filename to store the symmetric key (128 bit data) that was used for the inner wrapper. Valid only when specified symmetric algorithm is not null and --input-key-file is not specified. The TPM generates the key in this case.

? -C, --parent-context=OBJECT:

The parent key object.

? -U, --parent-public=FILE:

Specifies the file path to the public key of the parent object on the destination TPM. This should be a TPM2B_PUBLIC formatted file.

? -k, --private-key=FILE:

Specifies the file path to the external private key be encrypted for the remote TPM. This should be a PEM format private key.

? -r, --private=FILE:

Specifies the file path to save the private portion of the duplicated object. # Protection Details

Objects that can move outside of TPM need to be protected (confidentiality and integrity). For instance, transient objects require that TPM protected data (key or seal material) be stored outside of the TPM. This is seen in tools like tpm2_create(1), where the -r option outputs this protected data. This blob contains the sensitive portions of the object. The sensitive portions of the object are protected by the parent object, using the parent's symmetric encryption details to encrypt the sensitive data and HMAC it.

In-depth details can be found in sections 23 of:

? <https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.38.pdf>

Notably Figure 20, is relevant, even though it's specifically referring to duplication blobs, the process is identical.

If the output is from tpm2_duplicate(1), the output will be slightly different, as described fully in section 23.

? -u, --public=FILE:

Specifies the file path to save the public portion of the duplicated object, if an external key is being duplicated.

? -s, --encrypted-seed=FILE:

The file to save the encrypted seed of the duplicated object.

? -p, --auth=AUTH:

The authorization value for the key, optional.

? -L, --policy=FILE:

The input policy file, optional.

? -c, --key-context=OBJECT:

The object to be duplicated.

? --cphash=FILE

File path to record the hash of the command parameters. This is commonly termed as cpHash. NOTE: When this option is selected, The tool will not actually execute the command, it simply returns a cpHash.

References

Context Object Format

The type of a context object, whether it is a handle or file name, is determined according to the following logic in-order:

? If the argument is a file path, then the file is loaded as a restored TPM transient object.

? If the argument is a prefix match on one of:

? owner: the owner hierarchy

? platform: the platform hierarchy

? endorsement: the endorsement hierarchy

? lockout: the lockout control persistent object

? If the argument can be loaded as a number it will be treated as a handle, e.g. 0x81010013 and used directly._OBJECT_.

Authorization Formatting

Authorization for use of an object in TPM2.0 can come in 3 different forms: 1. Password 2. HMAC 3. Sessions

NOTE: Authorizations default to the EMPTY PASSWORD when not specified?.

Passwords

Passwords are interpreted in the following forms below using prefix identifiers.

Note: By default passwords are assumed to be in the string form when they do not have a prefix.

String

A string password, specified by prefix `?str:?` or its absence (raw string without prefix) is not interpreted, and is directly used for authorization.

Examples

```
foobar
```

```
str:foobar
```

Hex-string

A hex-string password, specified by prefix `?hex:?` is converted from a hexadecimal form into a byte array form, thus allowing passwords with non-printable and/or terminal un-friendly characters.

Example

```
hex:0x1122334455667788
```

File

A file based password, specified by prefix `?file:?` should be the path of a file containing the password to be read by the tool or a `?-?` to use stdin. Storing passwords in files prevents information leakage, passwords passed as options can be read from the process list or common shell history features.

Examples

```
# to use stdin and be prompted
```

```
file:-
```

```
# to use a file from a path
```

```
file:path/to/password/file
```

```
# to echo a password via stdin:
```

```
echo foobar | tpm2_tool -p file:-
```

```
# to use a bash here-string via stdin:
```

```
tpm2_tool -p file:- <<< foobar
```

Sessions

When using a policy session to authorize the use of an object, prefix the option argument with the session keyword. Then indicate a path to a session file that was created with `tpm2_startauthsession(1)`. Option? ally, if the session requires an auth value to be sent with the session handle (eg policy password), then append a + and a string as described in the Passwords section.

Examples

To use a session context file called `session.ctx`.

```
session:session.ctx
```

To use a session context file called `session.ctx` AND send the authvalue `mypassword`.

```
session:session.ctx+mypassword
```

To use a session context file called `session.ctx` AND send the HEX auth? value `0x11223344`.

```
session:session.ctx+hex:11223344
```

PCR Authorizations

You can satisfy a PCR policy using the `?pcr:?` prefix and the PCR mini? language. The PCR minilanguage is as follows:

```
<pcr-spec>=<raw-pcr-file>
```

The PCR spec is documented in in the section `?PCR bank specifiers?`.

The `raw-pcr-file` is an optional argument that contains the output of the raw PCR contents as returned by `tpm2_pcrread(1)`.

PCR bank specifiers (`pcr.md`)

Examples

To satisfy a PCR policy of sha256 on banks 0, 1, 2 and 3 use a specifi? er of:

```
pcr:sha256:0,1,2,3
```

specifying AUTH.

Algorithm Specifiers

Options that take algorithms support `?nice-names?`.

There are two major algorithm specification string classes, simple and complex. Only certain algorithms will be accepted by the TPM, based on

usage and conditions.

Simple specifiers

These are strings with no additional specification data. When creating objects, non-specified portions of an object are assumed to defaults.

You can find the list of known [Simple Specifiers Below](#).

Asymmetric

? rsa

? ecc

Symmetric

? aes

? camellia

Hashing Algorithms

? sha1

? sha256

? sha384

? sha512

? sm3_256

? sha3_256

? sha3_384

? sha3_512

Keyed Hash

? hmac

? xor

Signing Schemes

? rsassa

? rsapss

? ecdsa

? ecdaa

? ecschnorr

Asymmetric Encryption Schemes

? oaep

? rsaes

? ecdh

Modes

- ? ctr
- ? ofb
- ? cbc
- ? cfb
- ? ecb

Misc

- ? null

Complex Specifiers

Objects, when specified for creation by the TPM, have numerous algorithms to populate in the public data. Things like type, scheme and asymmetric details, key size, etc. Below is the general format for specifying this data: <type>:<scheme>:<symmetric-details>

Type Specifiers

This portion of the complex algorithm specifier is required. The remaining scheme and symmetric details will default based on the type specified and the type of the object being created.

- ? aes - Default AES: aes128
- ? aes128<mode> - 128 bit AES with optional mode (ctr|ofb|cbc|cfb|ecb).
If mode is not specified, defaults to null.
- ? aes192<mode> - Same as aes128<mode>, except for a 192 bit key size.
- ? aes256<mode> - Same as aes128<mode>, except for a 256 bit key size.
- ? ecc - Elliptical Curve, defaults to ecc256.
- ? ecc192 - 192 bit ECC
- ? ecc224 - 224 bit ECC
- ? ecc256 - 256 bit ECC
- ? ecc384 - 384 bit ECC
- ? ecc521 - 521 bit ECC
- ? rsa - Default RSA: rsa2048
- ? rsa1024 - RSA with 1024 bit keysize.
- ? rsa2048 - RSA with 2048 bit keysize.
- ? rsa4096 - RSA with 4096 bit keysize.

Scheme Specifiers

Next, is an optional field, it can be skipped.

Schemes are usually Signing Schemes or Asymmetric Encryption Schemes.

Most signing schemes take a hash algorithm directly following the signing scheme. If the hash algorithm is missing, it defaults to sha256.

Some take no arguments, and some take multiple arguments.

Hash Optional Scheme Specifiers

These scheme specifiers are followed by a dash and a valid hash algorithm, For example: oaep-sha256.

? oaep

? ecdh

? rsassa

? rsapss

? ecdsa

? ecschnorr

Multiple Option Scheme Specifiers

This scheme specifier is followed by a count (max size UINT16) then followed by a dash(-) and a valid hash algorithm. * ecdaa For example, ecdaa4-sha256. If no count is specified, it defaults to 4.

No Option Scheme Specifiers

This scheme specifier takes NO arguments. * rsaes

Symmetric Details Specifiers

This field is optional, and defaults based on the type of object being created and its attributes. Generally, any valid Symmetric specifier from the Type Specifiers list should work. If not specified, an asymmetric objects symmetric details defaults to aes128cfb.

Examples

Create an rsa2048 key with an rsaes asymmetric encryption scheme

```
tpm2_create -C parent.ctx -G rsa2048:rsaes -u key.pub -r key.priv
```

Create an ecc256 key with an ecdaa signing scheme with a count of 4 and

sha384 hash

```
/tpm2_create -C parent.ctx -G ecc256:ecdaa4-sha384 -u key.pub -r
```

```
key.priv cryptographic algorithms ALGORITHM.
```


This collection of options are common to many programs and provide information that many users may expect.

? -h, --help=[man|no-man]: Display the tools manpage. By default, it attempts to invoke the manpager for the tool, however, on failure will output a short tool summary. This is the same behavior if the ?man? option argument is specified, however if explicit ?man? is requested, the tool will provide errors from man on stderr. If the ?no-man? option is specified, or the manpager fails, the short options will be output to stdout.

To successfully use the manpages feature requires the manpages to be installed or on MANPATH, See man(1) for more details.

? -v, --version: Display version information for this tool, supported tctis and exit.

? -V, --verbose: Increase the information that the tool prints to the console during its execution. When using this option the file and line number are printed.

? -Q, --quiet: Silence normal tool output to stdout.

? -Z, --enable-errata: Enable the application of errata fixups. Useful if an errata fixup needs to be applied to commands sent to the TPM. Defining the environment TPM2TOOLS_ENABLE_ERRATA is equivalent. information many users may expect.

TCTI Configuration

The TCTI or ?Transmission Interface? is the communication mechanism with the TPM. TCTIs can be changed for communication with TPMs across different mediums.

To control the TCTI, the tools respect:

1. The command line option -T or --tcti
2. The environment variable: TPM2TOOLS_TCTI.

Note: The command line option always overrides the environment variable.

The current known TCTIs are:

? tabrmd - The resource manager, called tabrmd (<https://github.com/tpm2-software/tpm2-abrmd>). Note that tabrmd and

abrmd as a tcti name are synonymous.

? mssim - Typically used for communicating to the TPM software simulator.

tor.

? device - Used when talking directly to a TPM device file.

? none - Do not initialize a connection with the TPM. Some tools allow for off-tpm options and thus support not using a TCTI. Tools that do not support it will error when attempted to be used without a TCTI connection. Does not support ANY options and MUST BE presented as the exact text of ?none?.

The arguments to either the command line option or the environment variable are in the form:

<tcti-name>:<tcti-option-config>

Specifying an empty string for either the <tcti-name> or <tcti-option-config> results in the default being used for that portion respectively.

TCTI Defaults

When a TCTI is not specified, the default TCTI is searched for using dlopen(3) semantics. The tools will search for tabrmd, device and mssim TCTIs IN THAT ORDER and USE THE FIRST ONE FOUND. You can query what TCTI will be chosen as the default by using the -v option to print the version information. The ?default-tcti? key-value pair will indicate which of the aforementioned TCTIs is the default.

Custom TCTIs

Any TCTI that implements the dynamic TCTI interface can be loaded. The tools internally use dlopen(3), and the raw tcti-name value is used for the lookup. Thus, this could be a path to the shared library, or a library name as understood by dlopen(3) semantics.

TCTI OPTIONS

This collection of options are used to configure the various known TCTI modules available:

? device: For the device TCTI, the TPM character device file for use by the device TCTI can be specified. The default is /dev/tpm0.

Example: -T device:/dev/tpm0 or export TPM2TOOLS_TCTI=?device?

vice:/dev/tpm0?

? mssim: For the mssim TCTI, the domain name or IP address and port number used by the simulator can be specified. The default are 127.0.0.1 and 2321.

Example: -T mssim:host=localhost,port=2321 or export TPM2TOOLS_TC?

TI=?mssim:host=localhost,port=2321?

? abrmd: For the abrmd TCTI, the configuration string format is a series of simple key value pairs separated by a ';' character. Each key and value string are separated by a '=' character.

? TCTI abrmd supports two keys:

1. 'bus_name' : The name of the tabrmd service on the bus (a string).
2. 'bus_type' : The type of the dbus instance (a string) limited to 'session' and 'system'.

Specify the tabrmd tcti name and a config string of bus_name=com.ex?

ample.FooBar:

```
\--tcti=tabrmd:bus_name=com.example.FooBar
```

Specify the default (abrmd) tcti and a config string of bus_type=ses?

sion:

```
\--tcti:bus_type=session
```

NOTE: abrmd and tabrmd are synonymous. the various known TCTI modules.

EXAMPLES

To duplicate a key, one needs the key to duplicate, created with a policy that

allows duplication and a new parent:

```
tpm2_startauthsession -S session.dat
```

```
tpm2_policycommandcode -S session.dat -L policy.dat TPM2_CC_Duplicate
```

```
tpm2_flushcontext session.dat
```

```
tpm2_createprimary -C o -g sha256 -G rsa -c primary.ctx
```

```
tpm2_create -C primary.ctx -g sha256 -G rsa -r key.prv -u key.pub \
```

```
-L policy.dat -a "sensitiveorigin"
```

```
tpm2_loadexternal -C o -u new_parent.pub -c new_parent.ctx
```

```

tpm2_startauthsession --policy-session -S session.dat
tpm2_policycommandcode -S session.dat -L policy.dat TPM2_CC_Duplicate
tpm2_duplicate -C new_parent.ctx -c key.ctx -G null -p "session:session.dat" \
-r duprv.bin -s seed.dat
tpm2_flushcontext session.dat

```

As an end-to-end example, the following will transfer an RSA key generated on TPM-A to TPM-B

On TPM-B

Create a parent object that will be used to wrap/transfer the key.

```

tpm2_createprimary -C o -g sha256 -G rsa -c primary.ctx
tpm2_create -C primary.ctx -g sha256 -G rsa \
-r new_parent.prv -u new_parent.pub \
-a "restricted|sensitiveorigin|decrypt|userwithauth"

```

Copy new_parent.pub to TPM-A.

On TPM-A

Create root object and auth policy allows duplication only

```

tpm2_createprimary -C o -g sha256 -G rsa -c primary.ctx
tpm2_startauthsession -S session.dat
tpm2_policycommandcode -S session.dat -L dpolicy.dat TPM2_CC_Duplicate
tpm2_flushcontext session.dat
rm session.dat

```

Generate an RSA keypair on TPM-A that will be duplicated (note the passphrase is `foo`)

```

tpm2_create -C primary.ctx -g sha256 -G rsa -p foo -r key.prv \
-u key.pub -L dpolicy.dat -a "sensitiveorigin|userwithauth|decrypt|sign"
tpm2_load -C primary.ctx -r key.prv -u key.pub -c key.ctx
tpm2_readpublic -c key.ctx -o dup.pub

```

Test sign and encryption locally (so we can compare later that the same key was transferred).

```

echo "meet me at.." >file.txt
tpm2_rsaencrypt -c key.ctx -o data.encrypted file.txt
tpm2_sign -c key.ctx -g sha256 -f plain -p foo -o sign.raw file.txt

```

Compare the signature hash (we will use this later to confirm the key

was transferred to TPM-B):

```
sha256sum sign.raw
```

```
a1b4e3fbaa29e6e46d95cff498150b6b8e7d9fd21182622e8f5a3ddde257879e
```

Start an auth session and policy command to allow duplication

```
tpm2_startauthsession --policy-session -S session.dat
```

```
tpm2_policycommandcode -S session.dat -L dpolicy.dat TPM2_CC_Duplicate
```

Load the new_parent.pub file transferred from TPM-B

```
tpm2_loadexternal -C o -u new_parent.pub -c new_parent.ctx
```

Start the duplication

```
tpm2_duplicate -C new_parent.ctx -c key.ctx -G null \
```

```
-p "session:session.dat" -r dup.dpriv -s dup.seed
```

Copy the following files to TPM-B: * dup.pub * dup.dpriv * dup.seed *

(optionally data.encrypted just to test decryption)

On TPM-B

Start an auth,policy session

```
tpm2_startauthsession --policy-session -S session.dat
```

```
tpm2_policycommandcode -S session.dat -L dpolicy.dat TPM2_CC_Duplicate
```

Load the context we used to transfer

```
tpm2_flushcontext --transient-object
```

```
tpm2_load -C primary.ctx -u new_parent.pub -r new_parent.prv -c new_parent.ctx
```

Import the duplicated context against the parent we used

```
tpm2_import -C new_parent.ctx -u dup.pub -i dup.dpriv \
```

```
-r dup.prv -s dup.seed -L dpolicy.dat
```

Load the duplicated key context

```
tpm2_flushcontext --transient-object
```

```
tpm2_load -C new_parent.ctx -u dup.pub -r dup.prv -c dup.ctx
```

Test the imported key matches

? Sign

```
echo "meet me at.." >file.txt
```

```
tpm2_sign -c dup.ctx -g sha256 -o sig.rss -p foo file.txt
```

```
dd if=sig.rss of=sign.raw bs=1 skip=6 count=256
```

Compare the signature file hash:

```
$ sha256sum sign.raw
```

a1b4e3fbaa29e6e46d95cff498150b6b8e7d9fd21182622e8f5a3ddde257879e

? Decryption

```
tpm2_flushcontext --transient-object
tpm2_rsadecrypt -p foo -c dup.ctx -o data.ptext data.encrypted
# cat data.ptext
meet me at..
```

Exporting an OpenSSL RSA key for a remote TPM

To securely send an OpenSSL generated RSA key to a remote TPM such that only that remote TPM will be able to load it, and without exposing the private key to the host operating system on the remote machine:

? On the destination TPM-B, create a primary context and read its pub?

lic key, then send primary.pub to the source machine:

```
tpm2_createprimary -c primary.ctx
tpm2_readpublic -c primary.ctx -o primary.pub
```

? On the source machine create the RSA private key and wrap it for the destination TPM's public key. Similar to tpm2_makecredential, this step should not require a TPM.

```
openssl genrsa -out rsa.pem
tpm2_duplicate -U primary.pub -G rsa -k rsa.pem -u rsa.pub -r rsa.dpriv -s rsa.seed
```

? Send the rsa.pub, rsa.dpriv and rsa.seed to the destination TPM-B and

import the files, which will decrypt them using the primary.ctx to produce rsa.priv, which can then be loaded and used as a TPM key:

```
tpm2_import -C primary.ctx -G rsa -i rsa.dpriv -s rsa.seed -u rsa.pub -r rsa.priv
tpm2_load -C primary.ctx -c rsa.ctx -u rsa.pub -r rsa.priv
```

Returns

Tools can return any of the following codes:

- ? 0 - Success.
- ? 1 - General non-specific error.
- ? 2 - Options handling error.
- ? 3 - Authentication error.
- ? 4 - TCTI related error.
- ? 5 - Non supported scheme. Applicable to tpm2_testparams.

Github Issues (<https://github.com/tpm2-software/tpm2-tools/issues>)

HELP

See the Mailing List (<https://lists.01.org/mailman/listinfo/tpm2>)

tpm2-tools

tpm2_duplicate(1)