MSE | MASTER OF SCIENCE IN ENGINEERING

Trusted Platform Module (TPM)

Luca Haab

December 14, 2025

Trusted Platform Module (TPM)

A *Trusted Platform Module (TPM)* is a secure cryptoprocessor that implements the *ISO/IEC 11889* standard. Common uses are verifying that the boot process starts from a trusted combination of hardware and software and storing disk encryption keys.

— Wikipedia

TRUSTED® COMPUTING GROUP

Trusted Platform Module (TPM) was conceived by a computer industry consortium called *Trusted Computing Group (TCG)* that evolved into *ISO/IEC 11889:2015 — Trusted Platform Module Library Standard*

Cryptography: basics refresher (I)

Hash Functions

- A *hash function* H is a transformation that takes an input m and returns a fixed-size string, which is called the hash value h (that is, $h = H(m)$).
- The basic requirements for a cryptographic hash function are as follows:
  - The input can be of any length.
  - The output has a fixed length (*MD5*: 128 bits, *SHA-1*: 160 bits, ...)
  - $H(x)$ is relatively easy to compute for any given x
  - $H(x)$ is a one-way function
  - $H(x)$ is collision-free. That is, two distinct pieces of data share the same hash value

Example:

- *MD5* (must no longer be used)
- *SHA-1* (must no longer be used)
- *SHA-2*
- *SHA-3*
- *Blake2*
- Check https://csrc.nist.gov/projects/hash-functions for secure, up-to-date algorithms

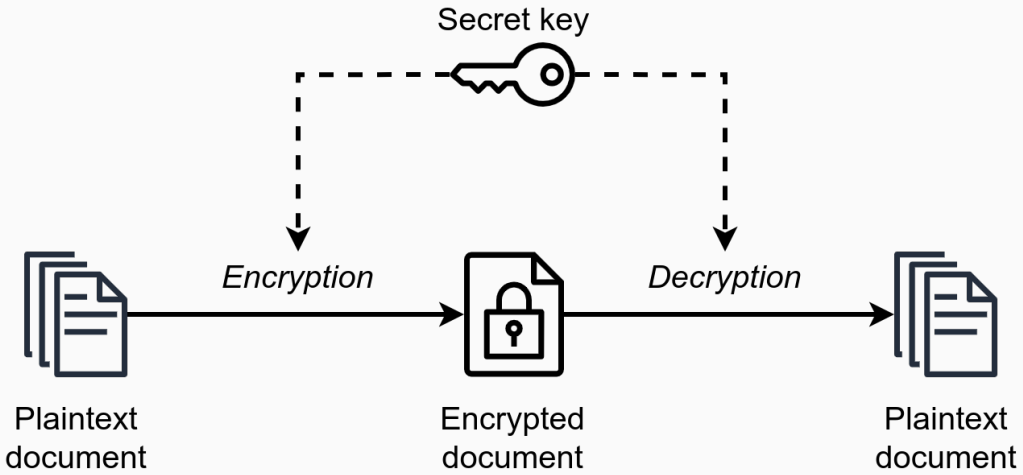
Source: *Konyaaltı, Antalya, Turkey March 2022 - Freshly squeezed pomegranate juice*, S. Darlin, Wikipedia

Cryptography: basics refresher (II)

Symmetric ciphers

A symmetric cipher main characteristic is that

- The same key is used for the encryption and decryption
- The security of this algorithm relies on the fact that only the sender and receiver of the message know the key
- It can ensure
  - data integrity
  - message authentication
  - non-repudiation
  - data confidentiality
- Common ciphers:
  - *AES*
  - *CHACHA20*
  - ...
- Check https://csrc.nist.gov/projects/block-cipher-techniques for secure, up-to-date algorithms (also Transitioning the Use of Cryptographic Algorithms and Key Lengths, NIST SP 800-175B Rev 1)



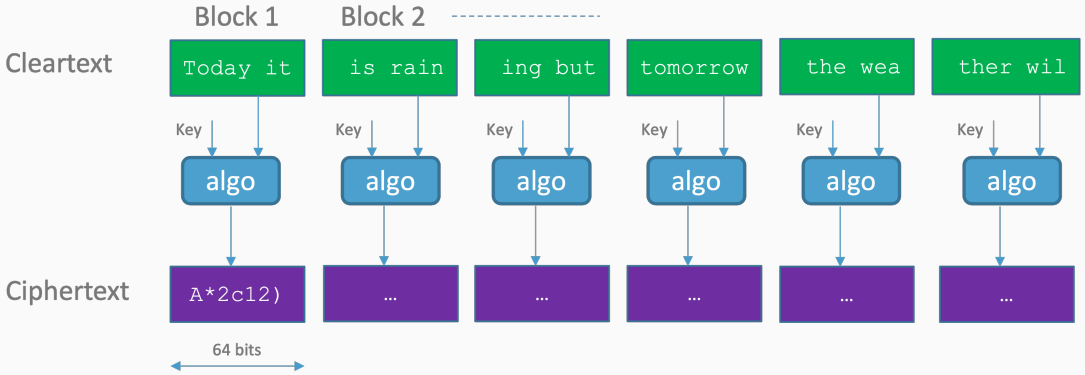
Source: *Symmetric Key Encryption*, Wikipedia

Cryptography: basics refresher (II)

Symmetric ciphers

- Cleartext is divided in blocks
- Each block has the same length (64, 128 bits)
- Each block is encrypted by an algorithm (*AES*, *IDEA*, *3DES*, ...) and a key

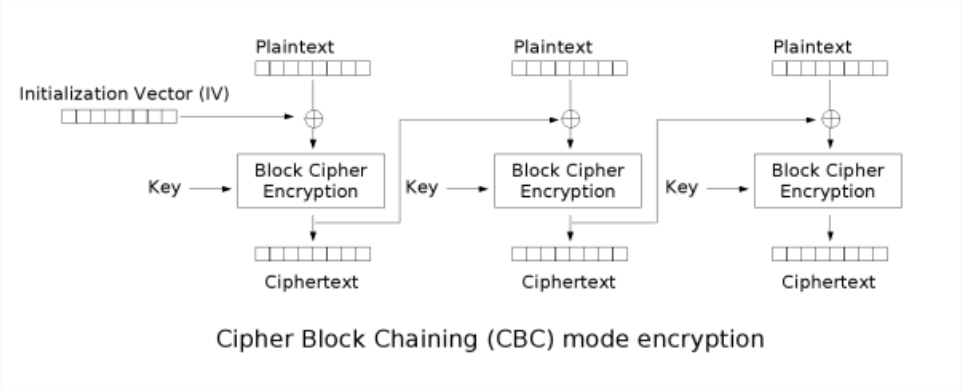
A concrete example: `Today it is raining but tomorrow the weather will be better`



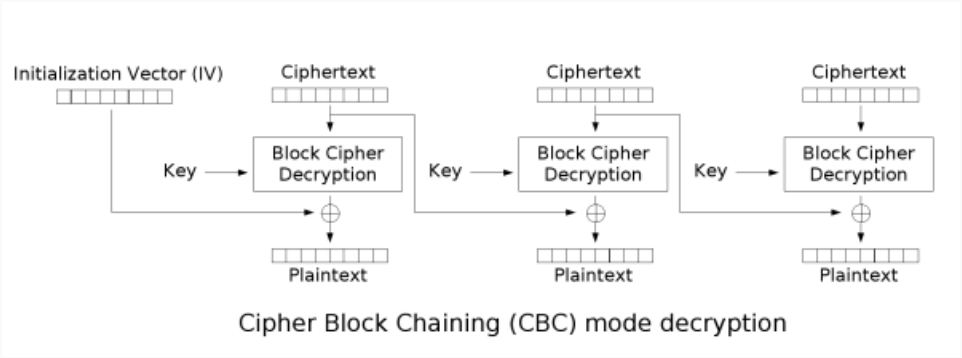
Cryptography: basics refresher (II)

A block cipher: *CBC (Cipher Block Chaining)*

- Every block is coded with the result of the previous block
- *IV: Initial Vector* for the first block. *IV* is not secret
- Interesting: *decryption* can be parallelized



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

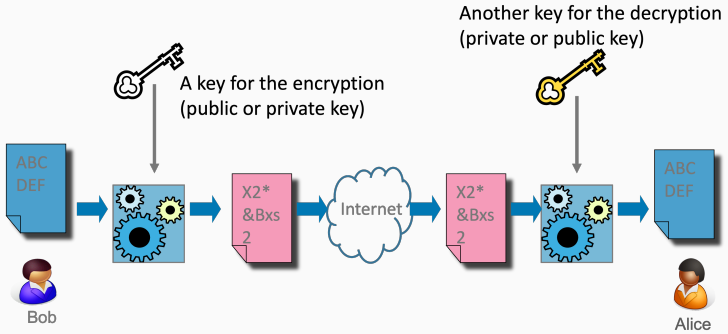
Source: *Cipher block chaining (CBC)*, Wikipedia

Cryptography: basics refresher (III)

Asymmetric ciphers

An asymmetric cipher main characteristics are :

- a pair of, mathematically linked, keys is used for the encryption and decryption (*public* and *private* keys)
- is also known as *public key encryption*
- encryption / decryption: asymmetric encryption uses the *public key* to *encrypt data* and the *private key* to *decrypt data*
- digital signatures: it enables the creation of *digital signatures*, which can be used to *verify the authenticity of data*
- secure key exchange: *asymmetric encryption* allows for *secure key exchange*, which is a critical feature in secure communication (e.g. *Diffie-Hellman key exchange algorithm* uses asymmetric encryption to establish a shared secret)



Cryptography: basics refresher (III)

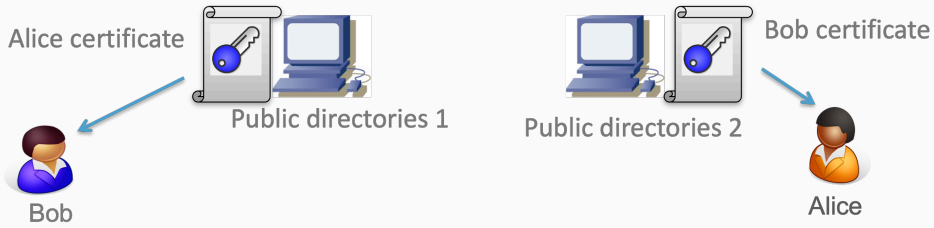
Asymmetric ciphers

Important aspects:

- two parties need to exchange their respective public keys
- generally the public keys are stored in a certificate. Alternatively, a directory service is used

TLS/SSL (*Transport Layer Security*)

PGP (*Pretty Good Privacy*)

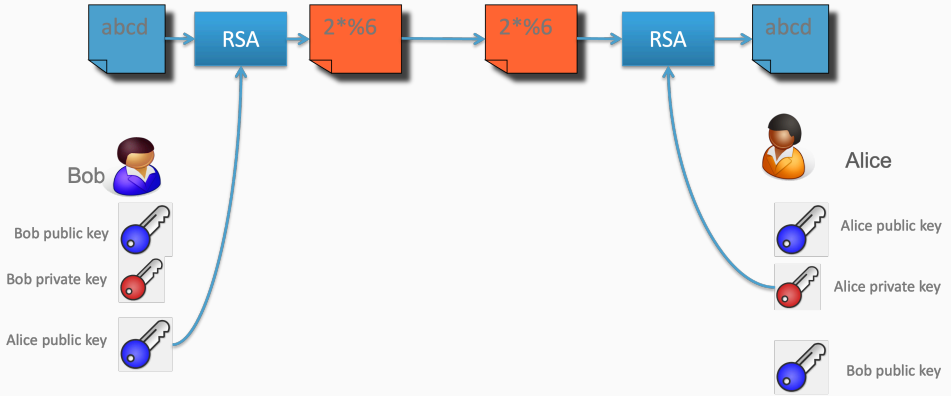


- Check *NIST Cryptographic Standards* for secure, up-to-date algorithms.

Usage of asymmetric ciphers

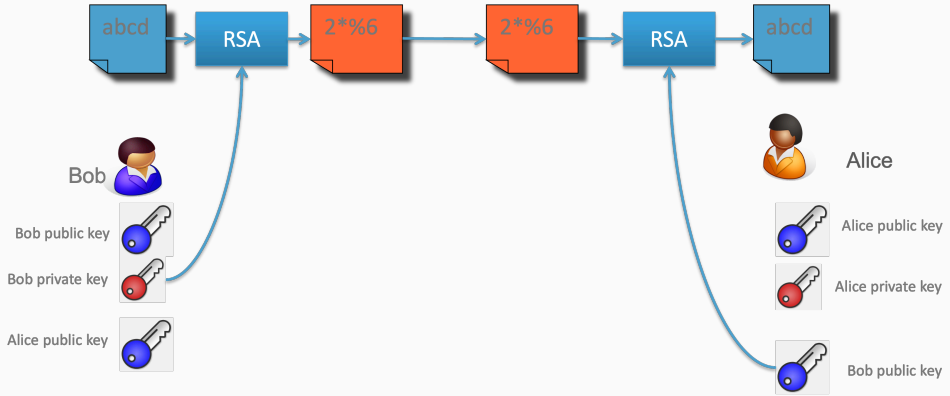
Encrypt with the public key and decrypt with the private key

- Confidentiality, integrity

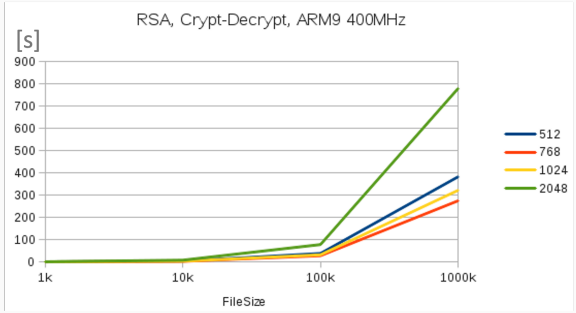


Encrypt with the private key and decrypt with the public key

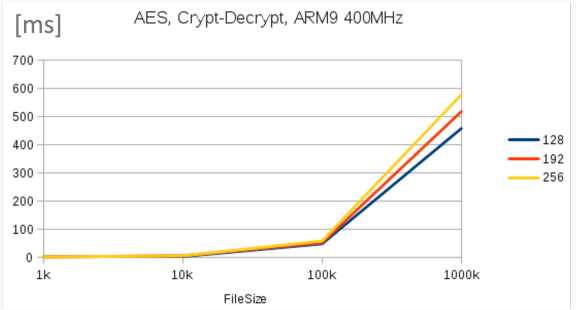
- Authenticity, integrity



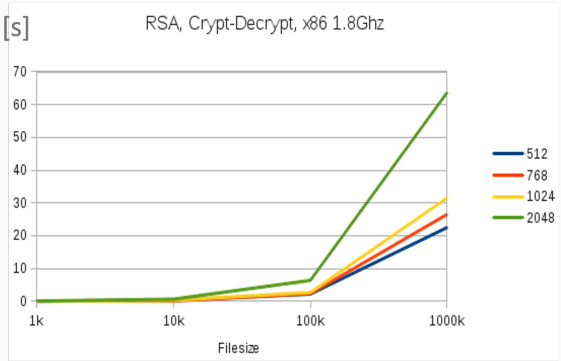
Cryptography Refresher: performance



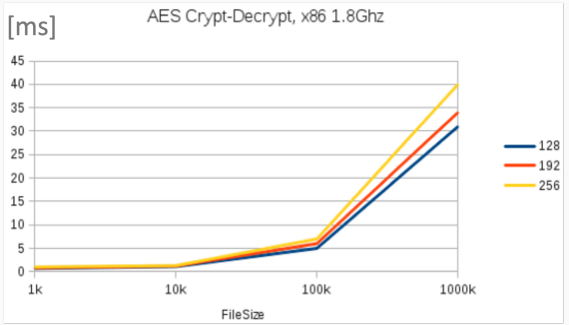
Asymmetric encryption on an ARM (ARM9, 400MHz) processor



Symmetric encryption on an ARM (ARM9, 400MHz) processor



Asymmetric encryption on an x86 (1.8GHz) processor



Symmetric encryption on an x86 (1.8GHz) processor

Digital Signature

A *Digital Signature* requires two steps:

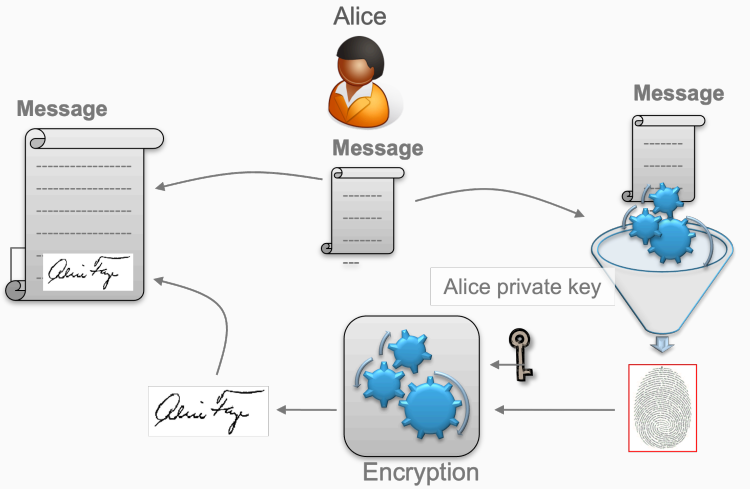
- a *hash* function that computes the message fingerprint
- the resulting *fingerprint* is encrypted by an *asymmetric cipher* using a *private key*

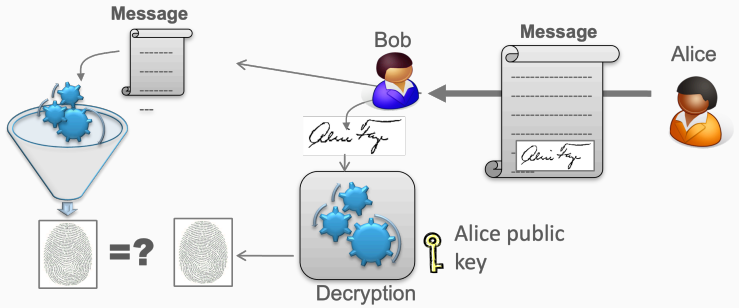
Goals of a *Digital Signature* are:

- checking the message integrity
- ensuring the origin-authentication of the message

In order to check a signature:

- The recipient, Bob, uses the sender's, Alice, *public key*
- Bob computes the message *fingerprint*
- Bob compares the two *fingerprints*





Cryptography Refresher: how to use openssl

- hash

```
md5sum file
  a6a0e8d0522e2c5de921b1c455506320
```

Or

```
openssl dgst -md5 file
MD5(file)= a6a0e8d0522e2c5de921b1c455506320
```

- symmetric

```
// Encrypt a file t.txt with the algorithm aes-256-cbc and save encrypted file to t.enc: IV and key are derived from a password
openssl enc -aes-256-cbc -e -in t.txt -out t.enc

// Decrypt a file t.enc with the algorithm aes-256-cbc and save decrypted file to t.txt
openssl enc -aes-256-cbc -d -in t.enc -out t.txt
```

- asymmetric

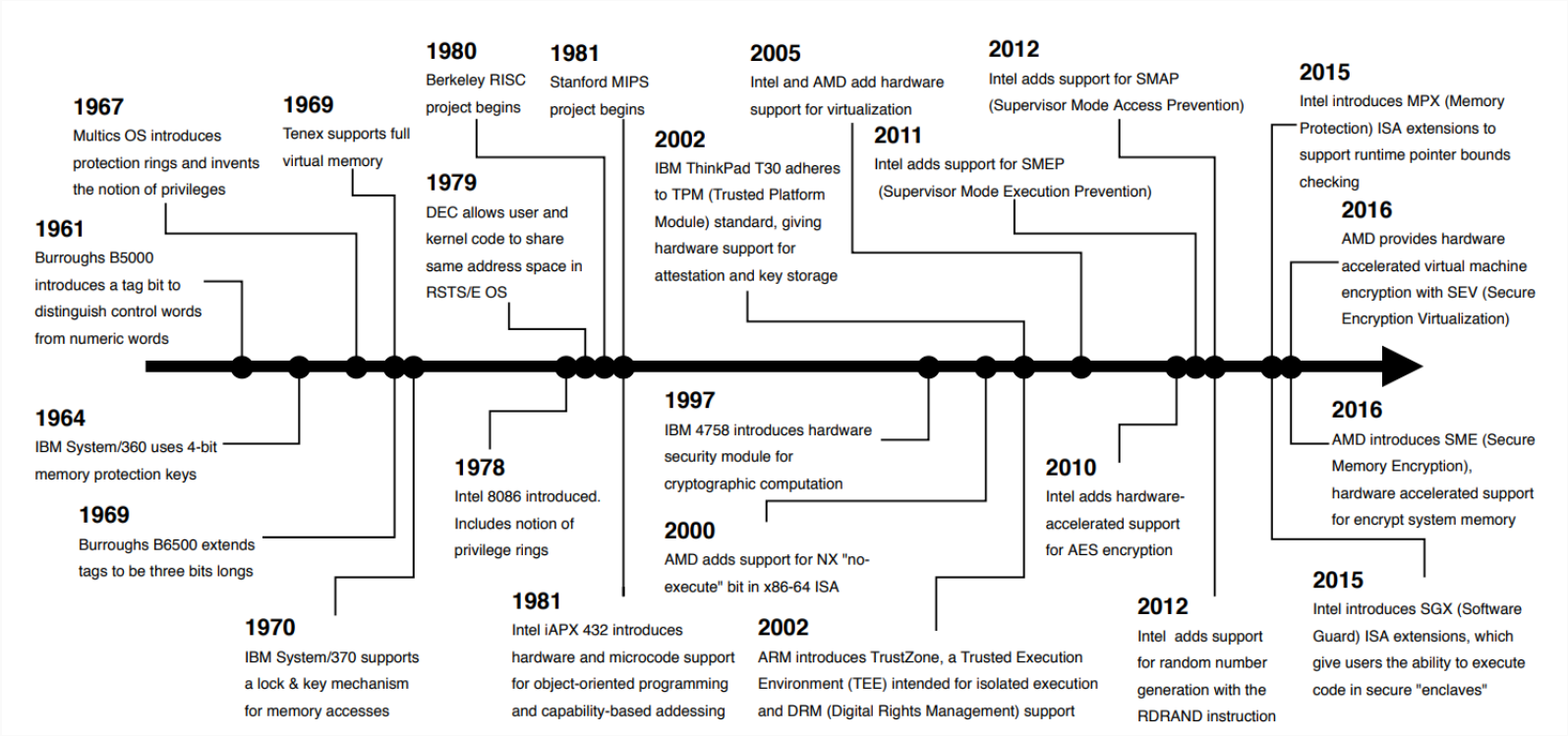
```
openssl genrsa -out rsa_key.pem 2048                    // compute private-public keys
openssl rsa -in rsa_key.pem -pubout -out rsa_key_pub.pem // get public key
openssl rsa -in rsa_key.pem -text                       // show private-public keys
openssl rsa -in rsa_key_pub.pem -pubin -text            // show public key

// encrypt t.txt with public key
openssl pkeyutl -encrypt -pubin -inkey rsa_key_pub.pem -in t.txt -out t.rsa

// decrypt t.rsa with private key
openssl pkeyutl -decrypt -inkey rsa_key.pem -in t.rsa -out t1.txt
```

Cryptography Refresher: how to use openssl

- digital signature

```
openssl genrsa -out rsa_key.pem 2048                    // compute private-public keys
openssl rsa -in rsa_key.pem -text                       // show keys
openssl rsa -in rsa_key.pem -pubout -out rsa_key_pub.pem // get public key

// sign t.txt with private key
openssl dgst -sha256 -sign rsa_key.pem -out t.sign t.txt

// verifyt.sign with public key
openssl dgst -verify -sha256 rsa_key_pub.pem -signature t.sign t.txt
```

Hardware Security Module (HSM)



Source: *Introduction to Security for Computer Architecture Students, Sixty years of hardware support for security*, A. Hastings, M. Tarek, S. Sethumadhavan, Columbia University, H&P Chapter 1 Supplement

Trusted Platform Module (TPM)

There exists two versions:

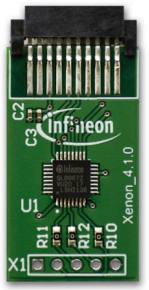
- Version 1.2: 2005, SHA-1, RSA with AES optional
  (officially became ISO/IEC 11889:2009 later)
- Version 2.0: 2014, SHA-256, RSA, ECC, AES
  with many other algorithms also defined but optional.
  (officially became ISO/IEC 11889:2015)

Important aspects:

- *TPM* is a passive device. It does not monitor the system and it cannot halt CPU execution. For it to work, it must be fed data
- *TPM* has limited storage for runtime state and persistent data: its non-volatile storage is small (~64KB), thus can only hold a limited number of objects at the same time
- *TPM* executes one command at a time, sequentially (cancelling is possible)

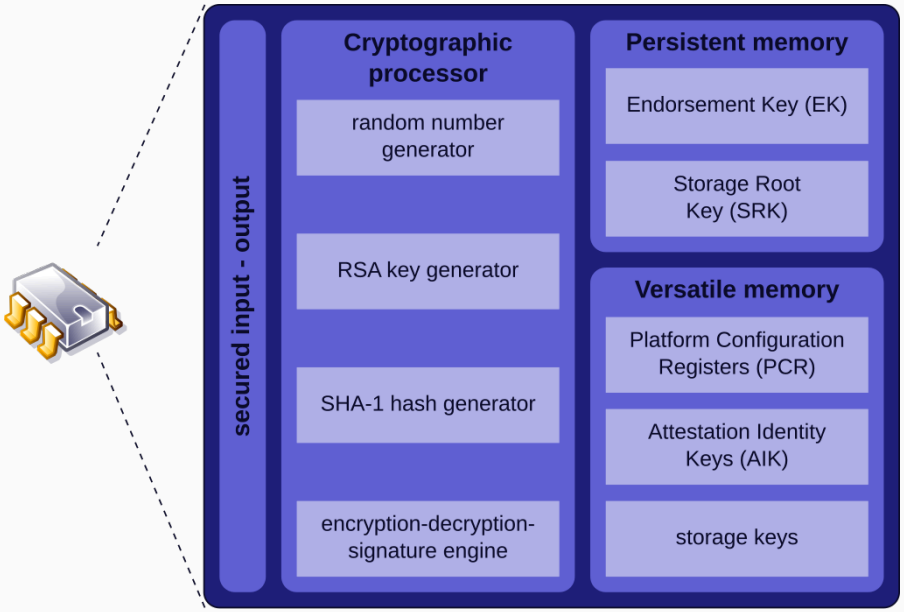
TRUSTED® COMPUTING GROUP

Trusted Platform Module (TPM) was conceived by a computer industry consortium called Trusted Computing Group (TCG) that evolved into *ISO/IEC 11889*:2015 — Trusted Platform Module Library Standard



Infineon OPTIGA TPM SLB 9672 KIT

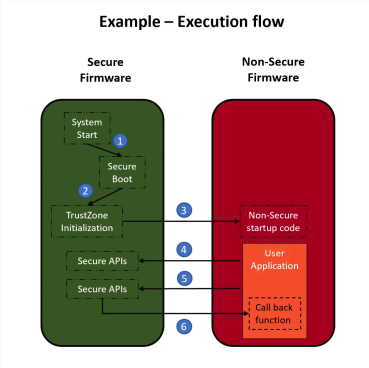
Trusted Platform Module (TPM)

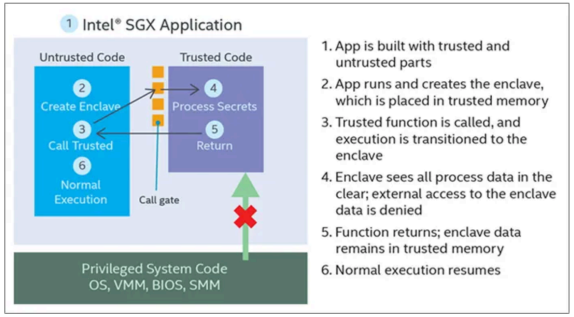


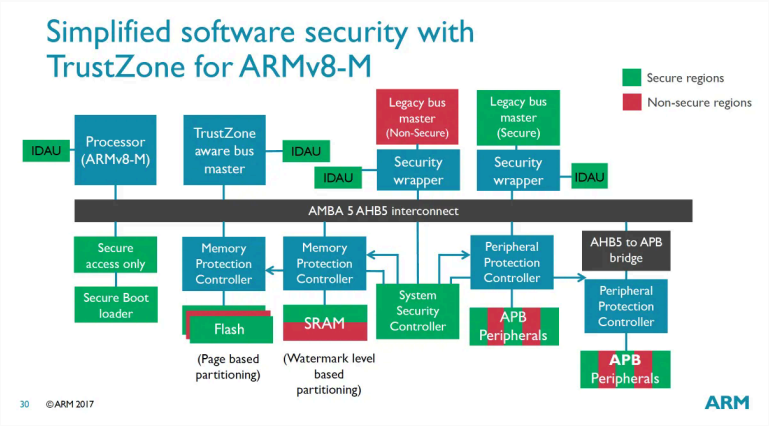
Source: *TPM 1.2 diagram : Wikipedia* (Author: G. Piolle)

TPM is not...

Be careful not to confuse other technologies (e.g. *ARM's Trustzone*, *Intel's SGX*) with the requirements of a *TPM*. These technologies do offer security features, but they relate to how the code is executed and the rights it does have. Obviously, one could combine these so as to enhance the security of ***TPM access*** but not as a replacement for it.







Sources: ARM and Demystifying ARM TrustZone for Microcontrollers - Medium

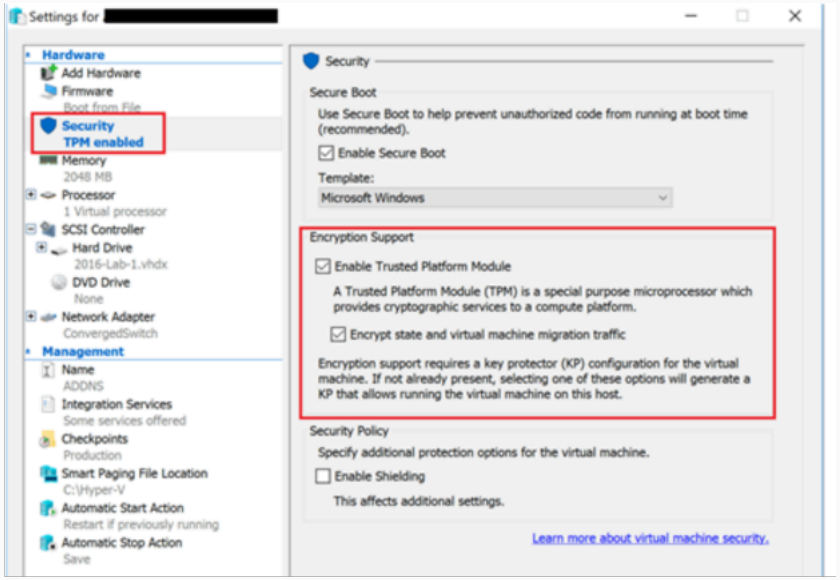
TPM Types

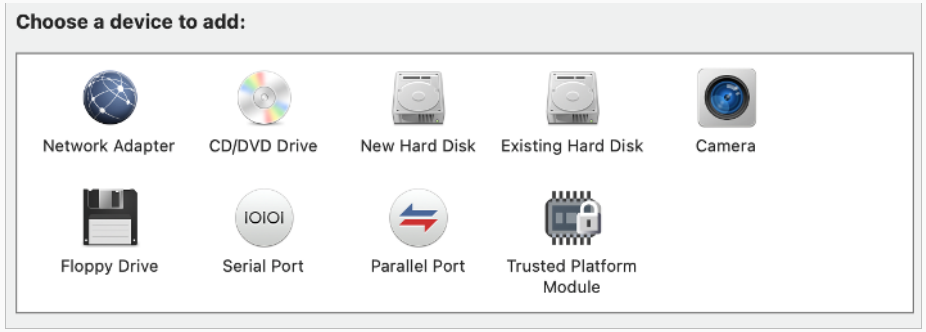
- ***DISCRETE TPM (dTPM)*** provides the highest level of security. The intent of this level is to ensure that the device it's protecting does not get hacked via even sophisticated methods. To accomplish this, a discrete chip is designed, built and evaluated for the highest level of security that can resist tampering with the chip, including probing it and freezing it with all sorts of sophisticated attacks.
- ***INTEGRATED TPM (iTPM)*** is the next level down in terms of security. This level still has a hardware TPM but it is integrated into a chip that provides functions other than security. The hardware implementation makes it resistant to software bugs, however, this level is not designed to be tamper-resistant.
- ***FIRMWARE TPM (fTPM)*** is implemented in protected software. The code runs on the main CPU, so a separate chip is not required. While running like any other program, the code is in a protected execution environment called a trusted execution environment (*TEE*) that is separated from the rest of the programs that are running on the CPU. By doing this, secrets like private keys that might be needed by the TPM but should not be accessed by others can be kept in the *TEE* creating a more difficult path for hackers. In addition to the lack of tamper resistance, the downside to the *TEE* or firmware TPM is that now the TPM is dependent on many additional aspects to keep it secure, including the *TEE* operating system, bugs in the application code running in the *TEE*, etc.
- ***SOFTWARE TPM*** can be implemented as a software emulator of the TPM. However, a software TPM is open to many vulnerabilities, not only tampering but also the bugs in any operating system running it. It does have key applications: it is very good for testing or building a system prototype with a TPM in it. For testing purposes, a software TPM could provide the right solution/approach.

TPM Types

Info

There exists the option of *vTPM*, which is a *TPM* offered by hypervisors. That is, it may leverage features like ARM Trustzone or Intel SGX for that.





TPM Types

TRUST ELEMENT	SECURITY LEVEL	SECURITY FEATURES	TYPICAL APPLICATION
DISCRETE TPM	HIGHEST	TAMPER RESISTANT HARDWARE	CRITICAL SYSTEMS
INTEGRATED TPM	HIGHER	HARDWARE	GATEWAYS
FIRMWARE TPM	HIGH	TEE	ENTERTAINMENT SYSTEMS
SOFTWARE TPM	NA	NA	TESTING & PROTOTYPING
VIRTUAL TPM	HIGH	HYPERVISOR	CLOUD ENVIRONMENT

TPM Certified List

TPM Certified Products

TCG Certified Programs | TNC Certified Products List | Storage Certified Products List

Search:

Company Name	Product Name	Product Revision	Specification	Details	Security Evaluation	Cert. Status	Cert. Complete Date
Nuvoton Technologies Corporation (NTC)	TPM NPCT76x	7.2.4.1	Version 2.0 - Revision 1.59		Complete	Complete	2025.10.2
Infineon Technologies	TPM SLB9672 TPM SLB9673	17.10, 17.12, 17.13 27.10, 27.13, 15.24, 16.24, 26.24, 17.24, 27.24	Version 2.0 - Revision 1.59		Complete	Complete	2025.01.0
Nuvoton Technologies Corporation (NTC)	TPM NPCT76x	7.2.5.0	Version 2.0 - Revision 1.59		Complete	Complete	2024.11.2
NSING Technologies	TPM NS350 V30	30.30	Version 2.0 - Revision 1.59	Errata 1.4	Complete	Complete	2024.11.1
STMicroelectronics	TPM ST33KTPM2XSPI TPM ST33KTPM2X	9.257	Version 2.0 - Revision 1.59	Errata 1.4	Complete	Complete	2024.10.0
STMicroelectronics	TPM ST33KTPM2I TPM ST33KTPM2A	10.257	Version 2.0 - Revision 1.59	Errata 1.4	Complete	Complete	2024.10.0
STMicroelectronics	TPM ST33KTPM2XSPI	9.258	Version 2.0 - Revision 1.59	Errata 1.4	Complete	Complete`	2024.10.0
Nuvoton Technologies Corporation (NTC)	TPM NPCT76x	7.2.4.0	Version 2.0 - Revision 1.59		Complete	Complete	2024.04.1
Infineon Technologies	TPM SLB9672 TPM SLB9673	16.10, 16.12, 16.13 26.10, 26.13	Version 2.0 - Revision 1.59		Complete	Complete	2023.08.0
Infineon Technologies	TPM SLB9672	15.20, 15.21, 15.22, 15.23	Version 2.0 - Revision 1.59		Complete	Complete	2023.07.0
STMicroelectronics	TPM ST33KTPM2XSPI TPM ST33KTPM2XI2C	9.256	Version 2.0 - Revision 1.59	Errata 1.3	Complete	Complete	2023.03.2

Source: *Trusted Platform Module (TPM) Certified Products*, Trusted Computing Group

TPM Use Cases

Classic TPM Use Case Categories

Use Case Category	Examples
Key Storage	Holds encryption keys, passwords, certificates and other sensitive information
Secure Authentication	Powers biometrics and other strong sign-in methods, ensuring credentials are not easily stolen
Enterprise Management	Device / Platform Attestation. That is, compliance measurement, remote trust verification and remote device health checks
Platform Integrity	Keep record of system boot data (firmware, drivers, OS, ...) allowing the system to verify it has not been tampered with
Storage Encryption	All data on a storage device (hard drive, SSD, etc.) is encrypted when "at rest" guaranteeing confidentiality of stored data

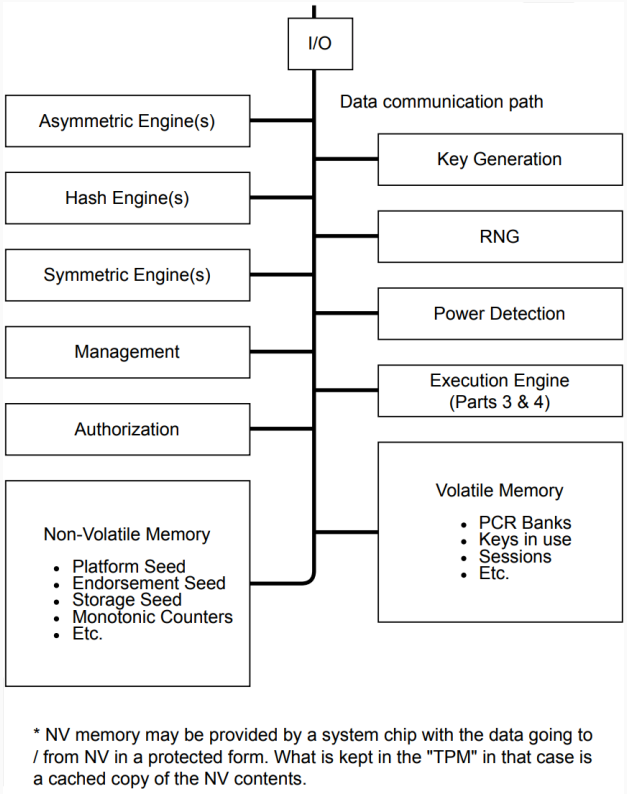
TPM Use Cases

NSA-Highlighted Use Cases (2024)

Use Case Category	Examples
Asset Management	TPM EK (Endorsement Key) / Platform Certificates used as hardware-rooted identifiers for inventory control and device tracking
Supply-Chain Auditing	Cryptographic verification of device provenance (through *Platform Certificates (PC)*), manufacturer provisioning and integrity from factory to deployment
Firmware Provenance	Vendor-signed manifests / delta-certificates to validate firmware state over device lifecycle
Full Software Supply-Chain	Using TPM measurement logs to validate runtime or software component integrity.

Based on: *Trusted Platform Module (TPM) Use Cases*, https://media.defense.gov/2024/Nov/06/2003579882/-1/-1/0/CSI-TPM-USE-CASES.PDF (Author: NSA)

TPM Architecture



Source: *Trusted Platform Module 2.0 Library Part 1: Architecture* (Author: TCG)

RNG: Random Number Generator
PCR: Platform Configuration Registers

Hashing
- SHA-1, SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512, ..)

Asymmetric Engine(s):
- RSA 1024/2048, ECC (Elliptic-Curve Cryptography)

Symmetric Engine(s):
- AES

Authorization:
- this subsystem is called by the Command Dispatch module at the beginning and end of command execution

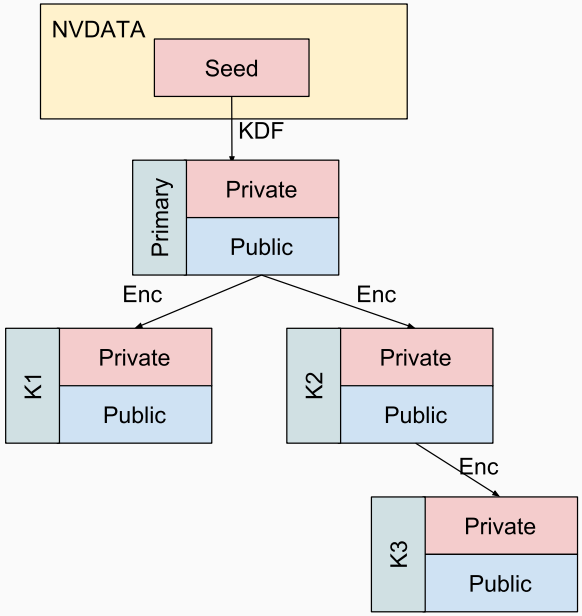
Info

There may be more algorithms, but they are not specified as mandatory.

TPM Cryptographic Keys

TPM generates strong, secure cryptographic keys.

- Strong in the sense that the key is derived from true random source and large key space. In case of 3DES, *TPM* also checks that keys are not known weak keys.
- Secure in the sense that the private key material never leaves the *TPM* secure boundary in plain form. When a key leaves the *TPM* - in order to be loaded and used later - it is wrapped (encrypted) by its parent key.



Keys form a tree: each child key is wrapped by its parent, all the way to the root of the tree, where the primary key is derived from a fixed seed. The seed is stored in the *TPM*'s *NVDATA* - and cannot be read externally.

TPM Primary key hierarchies

The *TPM* stores keys on one of four hierarchies:

- *Endorsement hierarchy*:

  The endorsement hierarchy is reserved for objects created and certified by the *TPM* manufacturer. The endorsement seed (*eseed*) is randomly generated at manufacturing time and never changes during the lifetime of the device. The primary endorsement key is certified by the *TPM* manufacturer, and because its seed never changes, it can be used to identify the device.

- *Platform hierarchy*:

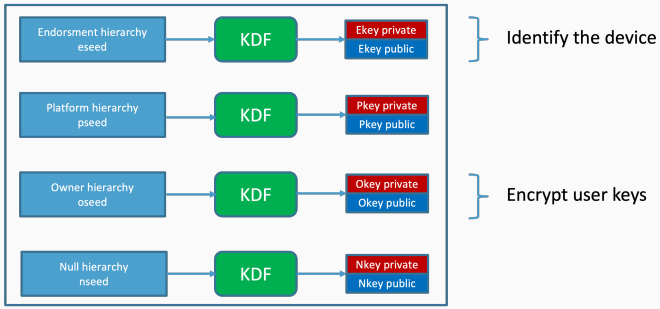
  The platform hierarchy is reserved for objects created and certified by the OEM that builds the host platform. The platform seed (*pseed*) is randomly generated at manufacturing time, but can be changed by the OEM by calling `tpm2_changepps`

- *Owner hierarchy*, also known as *storage hierarchy*:

  The owner hierarchy is reserved for end users - the primary users of the *TPM*. When a user takes a *TPM*, the owner hierarchy can be erased by the `tpm2_clear` command. `tpm2_clear` generates a new owner seed (oseed)

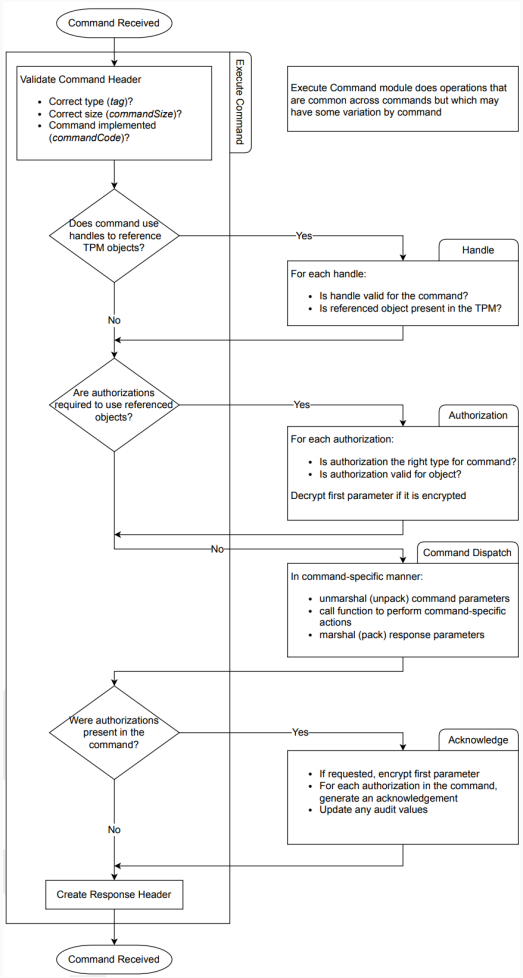
- *Null hierarchy*:

  The null hierarchy is reserved for ephemeral keys. The null seed is re-generated each time the host reboots



KDF: Key Derivation Function

TPM Command Execution flow



See *RESOURCES* for a bigger variant of the image.

Source of the image: [*Trusted Platform Module 2.0 Library Part 1: Architecture*] (Author: TCG)

TPM Examples

TPM Examples

Use Case	Description
1	Create primary keys
2	Save, show, delete keys to RAM, NVRAM
3	Get public keys from TPM
4	Create child asymmetric keys
5	Encrypt-decrypt with child asymmetric keys
6	Sign-verify with child asymmetric keys
7	Load an external public key, verify signature with external public key
8	Create symmetric key, encrypt-decrypt with symmetric key
9	Platform Configuration Registers (PCR) commands
10	Seal (save) data on TPM
11	Seal data on TPM and protect it with PCR policy

TPM commands

```
tpm2_getcap -l       : Display TPM capabilities
tpm2_flushcontext -t,
tpm2_flushcontext -s : Free a specified handle
tpm2_createprimary   : Create a primary key
tpm2_create          : Create a child object (key or sealing object)
tpm2_evictcontrol    : Transient to persistent or evict a persistent object
tpm2_load            : Load an object into the TPM
tpm2_loadexternal    : Load an external object into the TPM
tpm2_rsaencrypt,
tpm2_rsadecrypt,tpm2_sign,
tpm2_verifysignature,
tpm2_encryptdecrypt,
...
```

Info

Just make sure you use the `-h` option in case you have questions related to a command or use https://tpm2-tools.readthedocs.io/en/latest.

TPM commands

`tpm2_startauthsession`

Starts a session with the *TPM*. There are two types of policy session:

- *TPM_SE_TRIAL* (default), used for building a policy
- *TPM_SE_POLICY*, used for authenticating-authorizing, with a policy, the use of an object

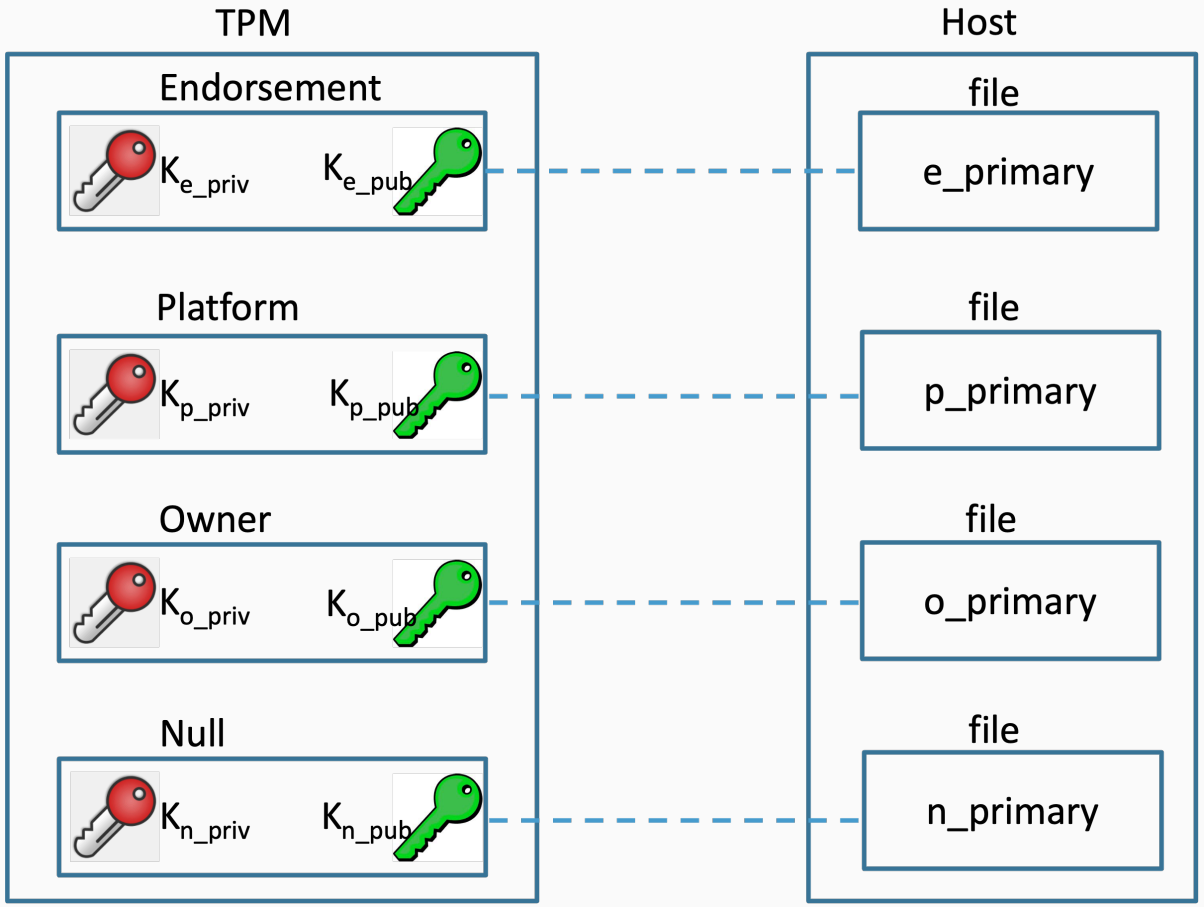
`tpm2_policypcr`

Generates a *PCR* (*Platform Configuration Registers*) policy event with the *TPM*. A *PCR policy event* creates a policy bound to specific PCR values and is useful within larger policies constructed using `policyor` and `policyauthorize` events.

```
tpm2_pcrextend  : Write (extend) PCR
tpm2_pcrread    : Read PCR
tpm2_pcrreset   : Reset PCR
```

> **Info**
>
> All the commands described will be used in examples on the next slides.

1. Primary Key Creation



1. Primary Key Creation

Primary Key Creation:

- Primary keys are derived from the primary *seeds* using a deterministic key derivation function (*KDF*). The template defines the characteristic of the primary key.
- The Private and Public keys are stored inside the TPM. The `primary.ctx` is a file saved outside the TPM. The `primary.ctx` contains reference to the primary keys

Concretely:

- Create RSA endorsement key:

  ```
  tpm2_createprimary -C e -G rsa2048 -c e_primary.ctx
  ```

- Create RSA platform key:

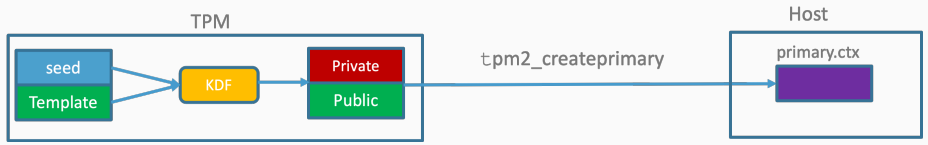
  ```
  tpm2_createprimary -C p -G rsa2048 -c p_primary.ctx
  ```

- Create RSA owner key:

  ```
  tpm2_createprimary -C o -G rsa2048 -c o_primary.ctx
  ```

- Create RSA null key:

  ```
  tpm2_createprimary -C n -G rsa2048 -c n_primary.ctx
  ```



Info

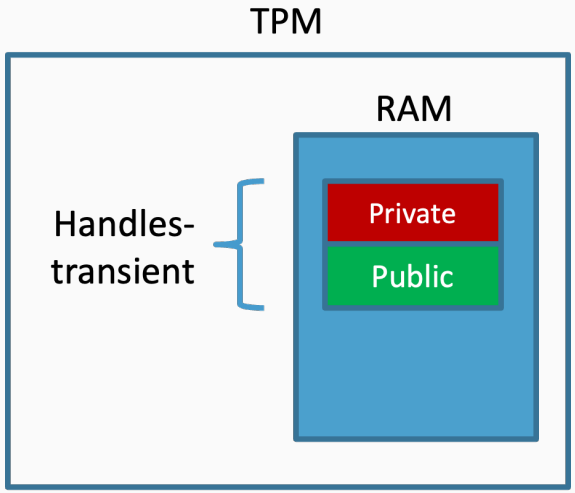
Make sure to use `man tpm2_createprimary` for additional information or go to https://tpm2-tools.readthedocs.io/en/latest.

1. Primary Key Creation

- After `tpm2_createprimary` command execution, public-private keys are stored in RAM in the transient area.
- `tpm2_getcap handles-transient` command shows the index of the public-private keys

Example with owner hierarchy:

```
tpm2_createprimary -C o -G rsa2048 -c o_primary.ctx
tpm2_getcap handles-transient
- 0x80000000
```



2. Save Keys To NVRAM

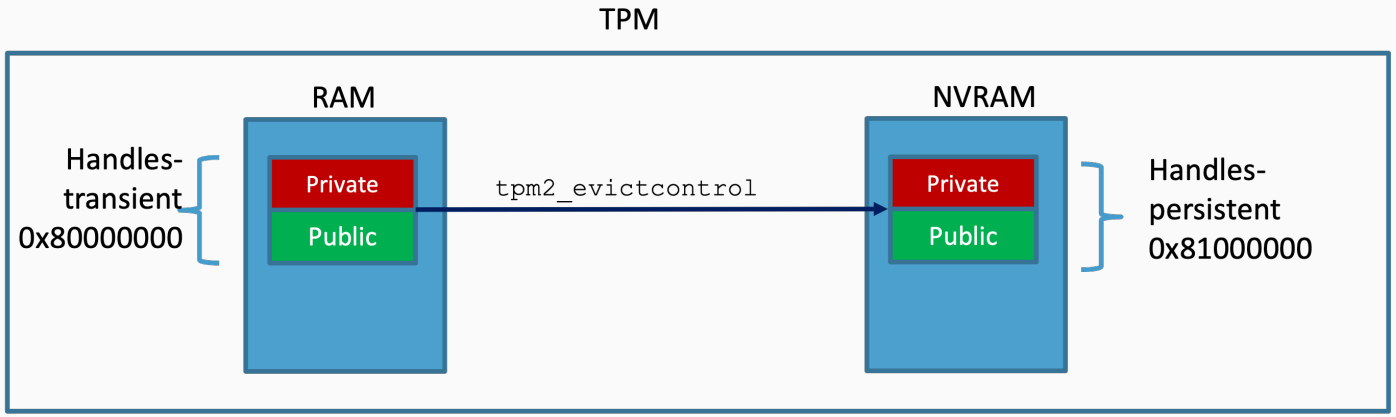
- It is possible to save these keys to NVRAM with the `tpm2_evictcontrol` command
- Keys are stored in the handles-persistent part of the TPM

```
tpm2_evictcontrol -c o_primary.ctx
```

or

```
tpm2_evictcontrol -c 0x80000000
```

```
tpm2_getcap handles-persistent
- 0x81000000
```



2. Delete Keys from NVRAM and RAM

- `tpm2_evictcontrol` command can store keys to *NVRAM* and erase keys in *NVRAM*

```
// Store key in NVRAM
tpm2_evictcontrol -c o_primary.ctx
```

or

```
tpm2_evictcontrol -c 0x80000000

tpm2_getcap handles-persistent
0x81000000

// Delete the key from NVRAM
tpm2_evictcontrol -c 0x81000000
```

- `tpm2_flushcontext` command deletes keys from *RAM*

```
tpm2_getcap handles-transient
- 0x80000000
- 0x80000001

tpm2_flushcontext 0x80000000
tpm2_flushcontext -t
```

3. Get Public Key from TPM

- `tpm2_readpublic` command get the public key from the TPM

```
tpm2_createprimary -C o -G rsa2048 -c o_primary.ctx
tpm2_getcap handles-transient
- 0x80000000

tpm2_readpublic -c o_primary.ctx --format PEM -o o_primary.public
```

Or

```
tpm2_readpublic -c 0x80000000 --format PEM -o o_primary.public
```

- The `openssl` command allows showing the public key

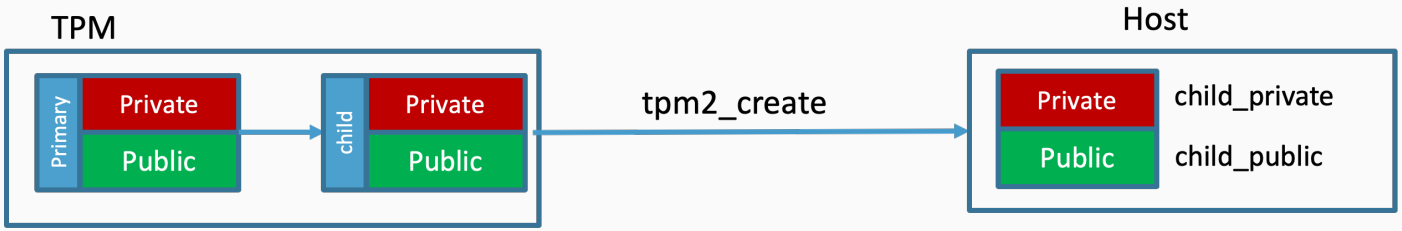
```
openssl rsa -pubin -in o_primary.public -text
```

4. Create Child Asymmetric Keys

`tpm2_create` command creates child objects (keys: asymmetric or symmetric, or sealing objects)

- `child_private` file contains the private asymmetric key or symmetric keys. These keys are encrypted by the parent keys.
- `child_public` file contains public asymmetric keys and different references

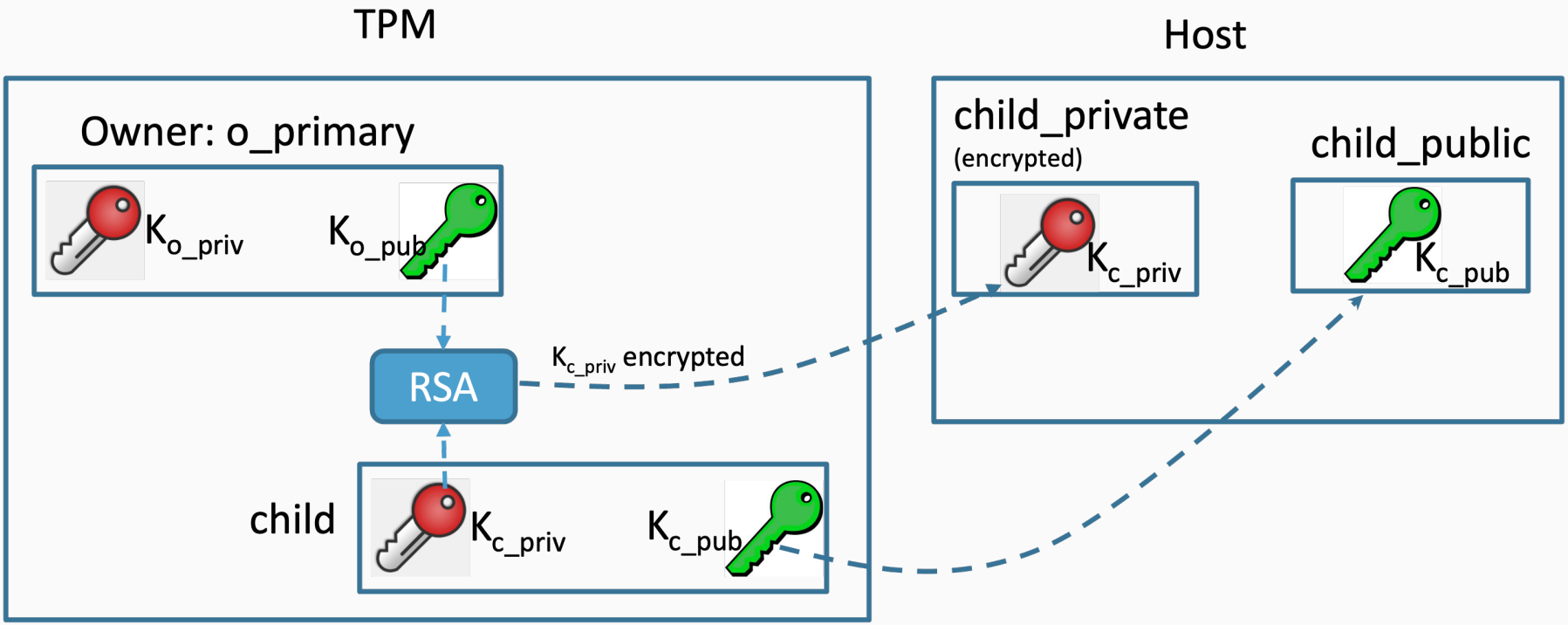
```
tpm2_createprimary -C o -G rsa2048 -c o_primary
tpm2_create -C o_primary -G rsa2048 -u child_public -r child_private
```



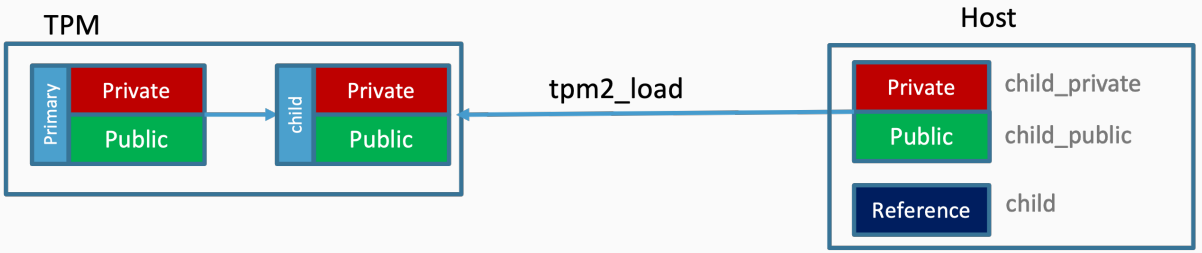
Info

Sealing is used by a TPM to secure user credentials and keys. See also *Infineon: Sealing and unsealing data in TPM*

4. Create Child Asymmetric Keys



4. Load Child Asymmetric Keys



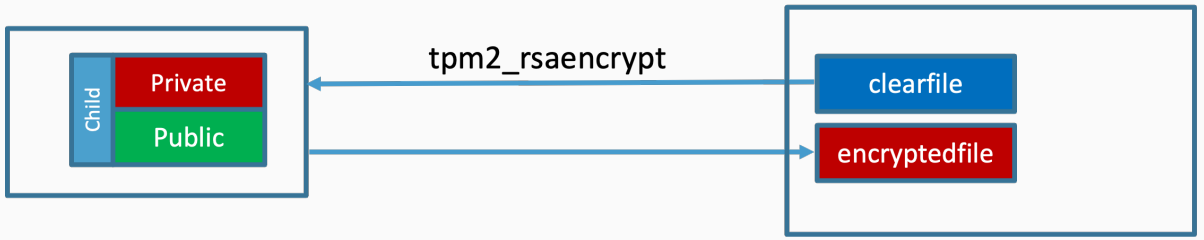
These child keys can be loaded to the TPM. The loaded keys are in the handles-transient area (RAM). The `child` file is a reference to the child keys

```
tpm2_createprimary -C o -G rsa2048 -c o_primary
tpm2_create -C o_primary -G rsa2048 -u child_public -r child_private
tpm2_load -C o_primary -u child_public -r child_private -c child
tpm2_getcap handles-transient        // child keys are in RAM (not NVRAM)
- 0x80000000
```

Save child keys to NVRAM with one of the following options

- `tpm2_evictcontrol -c child`
- `tpm2_evictcontrol -c 0x80000000`

5. RSA- Encrypt with Child Public Key




```
tpm2_createprimary -C o -G rsa2048 -c primary.ctx
tpm2_create -C primary.ctx -G rsa2048 -u child_public -r child_private
tpm2_load -C primary.ctx -u child_pub -r child_pr -c child

// encrypt with the public key
// meaning of -s flag (https://github.com/tpm2-software/tpm2-tools/blob/master/man/tpm2_rsaencrypt.1.md)
// -s for format: -s null: no padding, -s rsaes: pkcsv1.5 padding, -s oaep: oaep padding

tpm2_rsaencrypt -c child -s rsaes clearfile -o encryptedfile
```

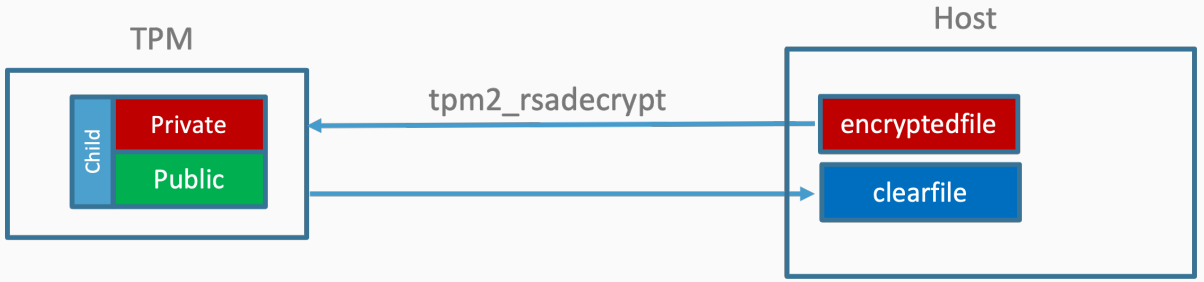
Info

Remark: the clearfile max length is equal the length of the key less the padding.

Examples for a *2048-bit key = 2048 ÷ 8 = 256 bytes*:

- null padding: *max plaintext=256 - 0 = 256 bytes*
- RSAES-PKCS#1 v1.5 (uses *11 bytes*): *max plaintext=256 - 11 = 245 bytes*

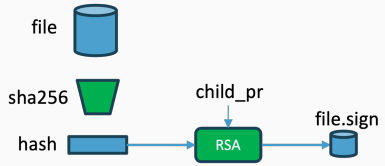
5. RSA- Decrypt with Child Public Key




```
tpm2_createprimary -C o -G rsa2048 -c primary.ctx
tpm2_create -C primary.ctx -G rsa2048 -u child_public -r child_private
tpm2_load -C primary.ctx -u child_pub -r child_pr -c child

// decrypt with the private key
tpm2_rsadecrypt -c child -s rsaes encryptedfile -o clearfile
```

6. RSA- Sign with Child Private Key

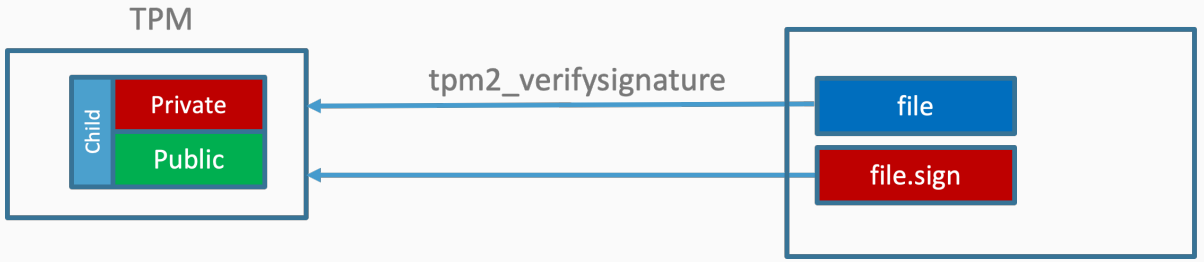

```
tpm2_createprimary -C o -G rsa2048 -c primary.ctx
tpm2_create -C primary.ctx -G rsa2048 -u child_public -r child_private
tpm2_load -C primary.ctx -u child_pub -r child_pr -c child


tpm2_sign -c child -g sha256 -o file.sign file //sign with the private key
```



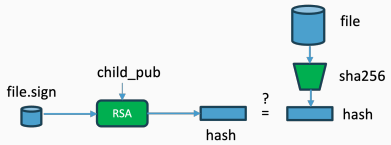
Warning

Remark: The message hash is computed by the TPM (be careful if the message is big).

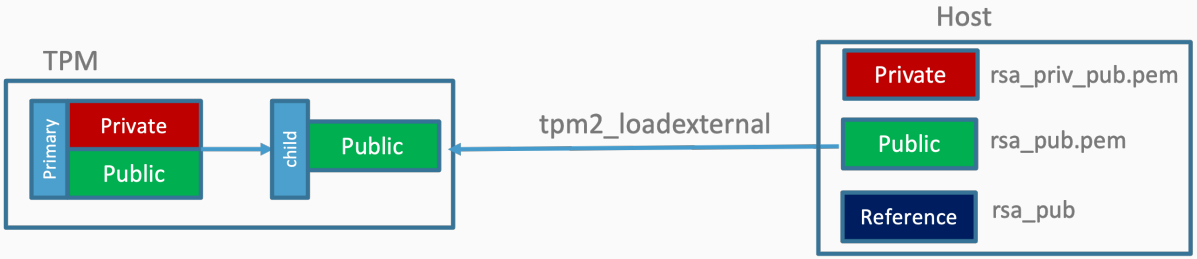
6. RSA- Verify Signature with Child Keys




```
tpm2_createprimary -C o -G rsa2048 -c primary.ctx
tpm2_create -C primary.ctx -G rsa2048 -u child_public -r child_private
tpm2_load -C primary.ctx -u child_pub -r child_pr -c child

// Verify with the private key
tpm2_verifysignature -c child -g sha256 -s file.sign -m file
```



7. Load an External Public Key



It is possible to import external public keys to the TPM. Generate rsa public key with openssl (`rsa_priv_pub.pem` contains private and public keys, `rsa_pub.pem` contains only public key)

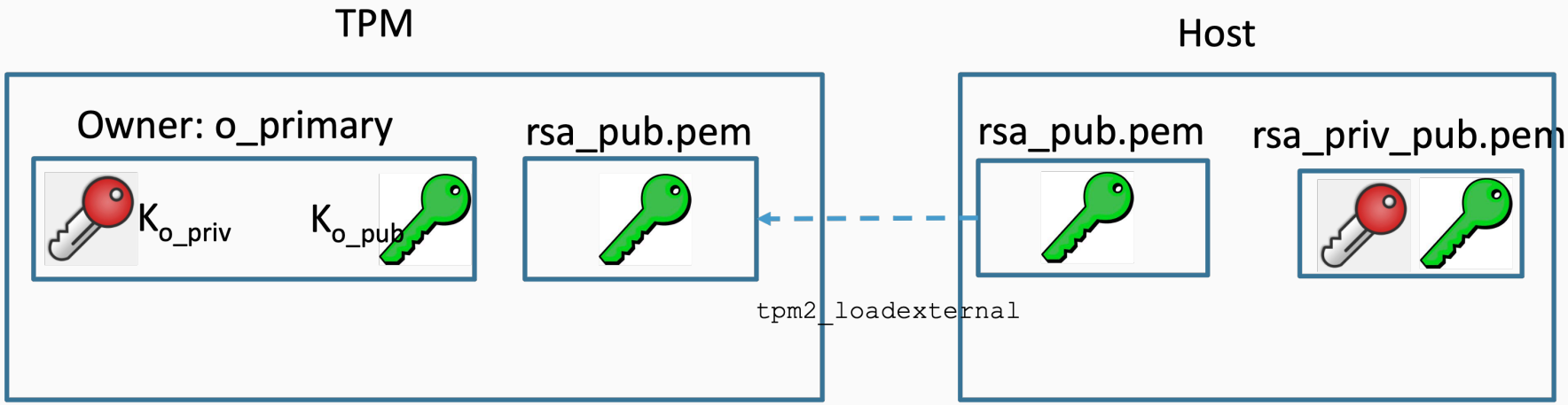
```
openssl genrsa -out rsa_priv_pub.pem 2048
openssl rsa -in rsa_priv_pub.pem -out rsa_pub.pem -pubout
```

`tpm2_loadexternal` command loads `rsa_pub.pem` into the TPM the owner hierarchy. The key is in the handles-transient area.

The `rsa_pub` file contains references to the loaded key (used for the next commands)

```
tpm2_loadexternal -C o -G rsa -u rsa_pub.pem -c rsa_pub
```

7. Load an External Public Key



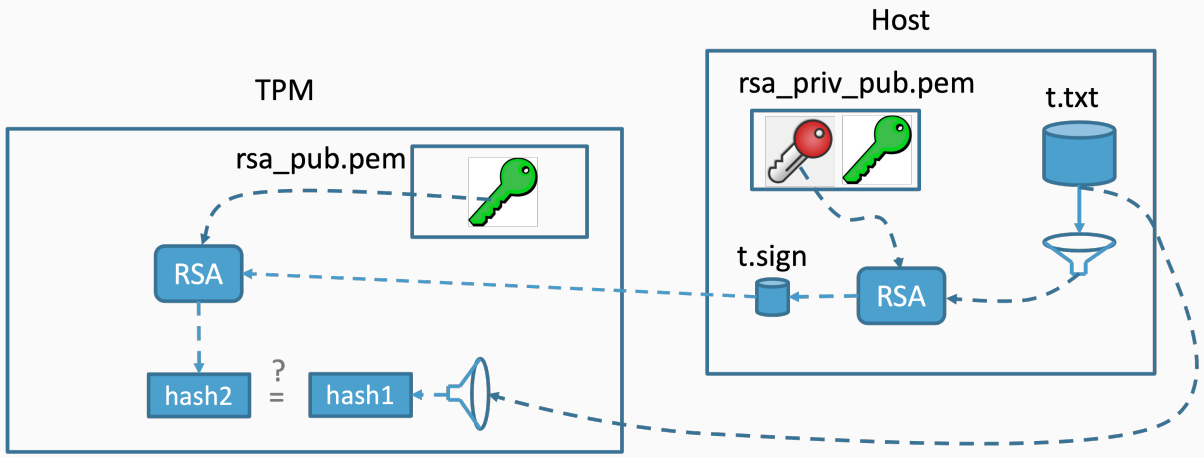
7. Verify Signature with an External Public Key

On the Host, sign a file with the private key

```
openssl dgst -sha256 -sign rsa_priv_pub.pem -out t.sign t.txt
```

On the TPM, verify the signature with the public key

```
tpm2_loadexternal -C o -G rsa -u rsa_pub.pem -c rsa_pub
tpm2_verifysignature -c rsa_pub -g sha256 -f rsassa -s t.sign -m t.txt
```



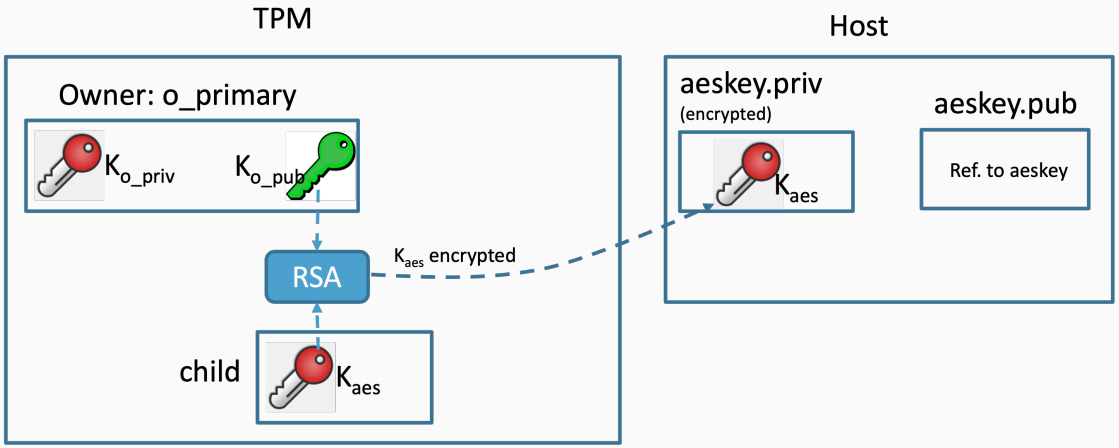
8. Create Child Symmetric Key

`tpm2_create` command creates an AES symmetric key (K_{aes}). K_{aes} is encrypted by primary public key K_{o_pub} and saved to `aeskey.priv`.

- `aeskey.priv` contains the K_{aes} symmetric key, this key is encrypted ith the K_{o_pub} key
- `aeskey.pub` contains some references to K_{aes} key

Next slide:

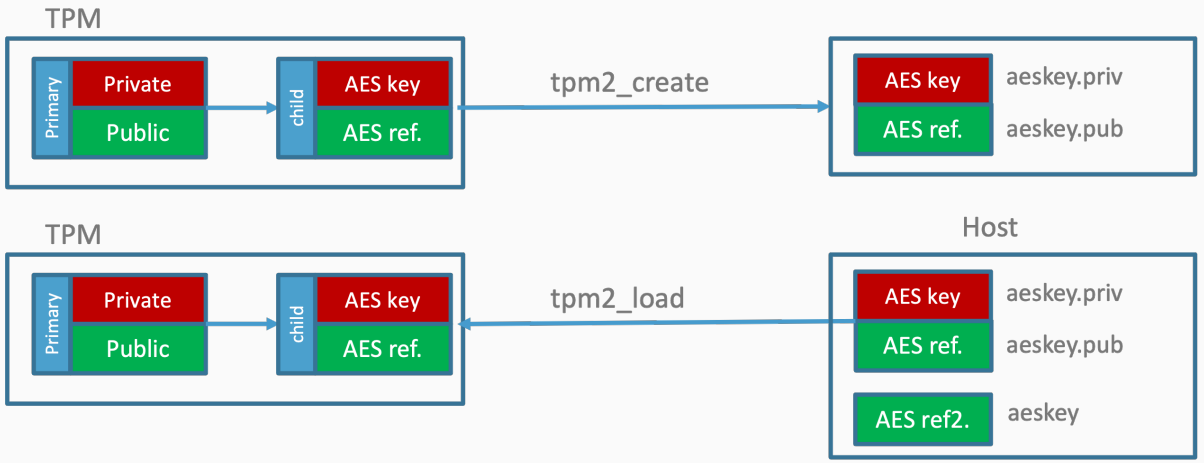
- K_{aes} is loaded to the TPM with `tpm2_load`, aeskey contains references to K_{aes}
- In order to encrypt-decrypt, it is necessary to have an *Initial Vector* with 16 bytes



56

8. Symmetric Encryption-Decryption

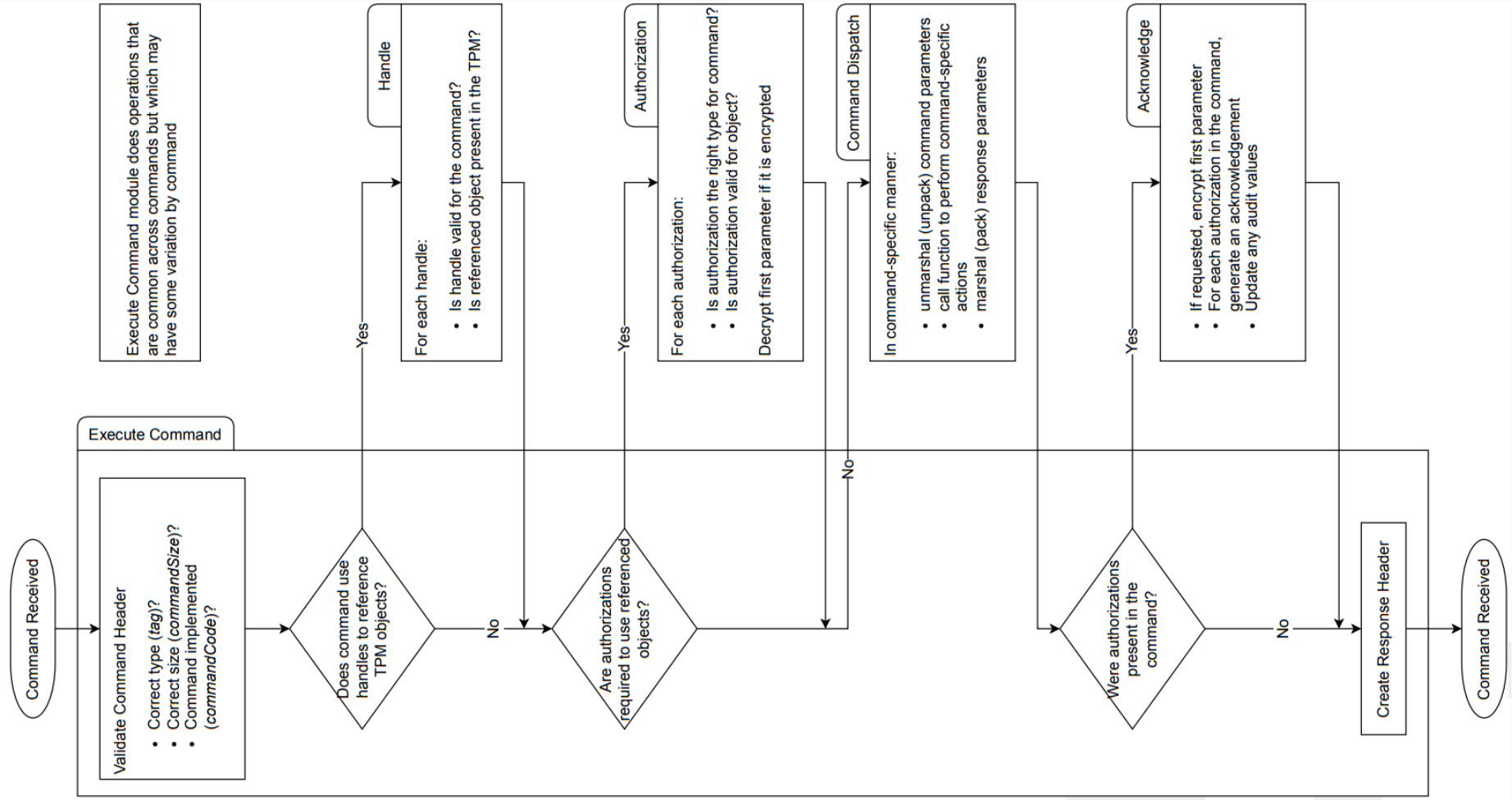
```
tpm2_createprimary -C o -G rsa2048 -c primary
tpm2_create -C primary -G aes128cbc -u aeskey.pub -r aeskey.priv
tpm2_load -C primary -u aeskey.pub -r aeskey.priv  -c aeskey

echo -n 234sdfweaw34rft6 > IV   // 16 bytes

tpm2_encryptdecrypt -c aeskey -e --iv=IV --pad -o encryptedfile clearfile
tpm2_encryptdecrypt -c aeskey -d --iv=IV -o clearfile encryptedfile
```



Resources

- A Practical Guide to TPM 2.0 - Using the Trusted Platform Module in the New Age of Security: https://link.springer.com/book/10.1007/978-1-4302-6584-9
- TPM 2.0 Library: https://trustedcomputinggroup.org/resource/tpm-library-specification/
- *Google's TPM in a browser*: https://google.github.io/tpm-js/#pg_welcome
- *NIST Cryptographic Standards*: https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines

Resources


Command Received
Execute Command
Validate Command Header
• Correct type (tag)?
• Correct size (commandSize)?
• Command implemented (commandCode)?
Does command use handles to reference TPM objects?
Yes
No
Handle
For each handle:
• Is handle valid for the command?
• Is referenced object present in the TPM?
Are authorizations required to use referenced objects?
Yes
No
Authorization
For each authorization:
• Is authorization the right type for command?
• Is authorization valid for object?
Decrypt first parameter if it is encrypted
Command Dispatch
In command-specific manner:
• unmarshal (unpack) command parameters
• call function to perform command-specific actions
• marshal (pack) response parameters
Were authorizations present in the command?
Yes
No
Acknowledge
• If requested, encrypt first parameter
• For each authorization in the command, generate an acknowledgement
• Update any audit values
Create Response Header
Command Received
Execute Command module does operations that are common across commands but which may have some variation by command