luna-kmu User Manual

Version 0.1.0

`kmu` is a key management CLI for Thales Luna HSMs: generate and manage keys, encrypt and decrypt, move key material in and out, and negotiate a key with a payShield TMD.

Every command runs once and holds no session. The full parameter list for any command is always available from `kmu <command> --help`.

Contents

1. Getting started 4
Getting the binary 4
Prerequisites 4
Your first commands 4
Global options 4
Exit codes 5
2. Connecting and authenticating 6
Four ways to supply the PIN 6
Roles 6
Verifying credentials 6
3. Inspection commands 7
List slots 7
List objects 7
Show one object's attributes 7
Show supported mechanisms 8
4. Generating keys 9
Symmetric keys 9
Asymmetric key pairs 9
Attribute flags 10
Clear key components 10
5. Object management 11
Create a data object 11
Modify attributes 11
Read and write a single attribute 11
Digest and KCV 12
Delete 12
The four KCV methods 12
6. Cryptographic operations 14

Encrypt and decrypt 14
Key derivation 14
7. Export and import 16
Public keys 16
Symmetric and private keys (wrapped under another key) 16
Password-protected private keys (PKCS#8 PBES2) 16
TR-31 key blocks 17
Clear-component import 17
File encoding conversion 17
8. Negotiating a key with a payShield TMD 19
9. Troubleshooting 20
`CKR_KEY_NOT_WRAPPABLE` 20
`CKR_OPERATION_NOT_ALLOWED` from `digestkey` 20
KCV does not match another tool 20
`CKR_ECC_UNKNOWN_CURVE` 20
OpenSSL cannot read a PKCS#8 file 20
Entering components and passwords in a pipeline 20
PKCS#11 library not found 21

1. Getting started

Getting the binary

`kmu` is a single executable. No installation, no extra runtime:

Platform	File
Linux x86-64	`kmu`
Windows x86-64	`kmu.exe`

Copy it anywhere and run it. Put it on your `PATH` and you can type `kmu` directly, as every example below assumes; otherwise call it by full path.

Prerequisites

- Luna Client installed with `/etc/Chrystoki.conf` (Windows: `C:\Program Files\SafeNet\LunaClient\crystoki.ini`) configured — kmu reads the PKCS#11 library location from its `Chrystoki2` section, **or**
- `ChrystokiConfigurationPath` pointing at the **directory** holding that file (pointing at the file itself, or at a `.so`/`.dll`, is an error), **or**
- the PKCS#11 library path given via `--library`, which bypasses the config file

Your first commands

```
# See which slots exist
kmu listslot

# Put the PIN in an environment variable — not on the command line
export HSM_PIN='your-pin'

# List the objects in slot 0
kmu list --slot 0 --pin-env HSM_PIN
```

Global options

Option	Meaning
`--library <path>`	PKCS#11 library. Falls back to `ChrystokiConfigurationPath`
`--json`	Emit JSON instead of human-readable output, for scripting
`-h, --help`	Show help (available on every subcommand)
`-V, --version`	Show the version and the product URL

```
kmu --help              # command overview
kmu export --help       # full parameters for one command
kmu help export         # same thing
```

The first line of the help is a title chosen by the system locale: with a Chinese locale (language code `zh`, Traditional or Simplified) it reads "Luna KMU 命令列版 版權所有 (c) 雲杉科技"; with any other locale, "Luna KMU CLI Copyright (c) Spruce Technology Co., Ltd. Taiwan.". The second line is the product URL, https://sprucets.com/tools/luna-kmu/, which `-V` prints as well. On Linux the locale comes from `LANGUAGE`, `LC_ALL`, `LC_MESSAGES` and `LANG`, in that order; on Windows from the user's display language. The rest of the help text is in Chinese regardless.

Exit codes

Code	Meaning
0	Success
1	Runtime error (object not found, HSM returned an error, …)
2	Invalid argument (including PKCS#11 library configuration errors)
3	Connection or login failure

2. Connecting and authenticating

Four ways to supply the PIN

Pick one (giving two is an argument error); with none of them you are prompted:

```
kmu list --slot 0 --ped                    # 1. PED authentication, no PIN sent
kmu list --slot 0 --pin-env HSM_PIN        # 2. From an environment variable (recommended)
kmu list --slot 0 --pin-file /run/pin.txt  # 3. From a file (trailing newline stripped)
kmu list --slot 0 --pin 'your-pin'         # 4. Inline (discouraged)
kmu list --slot 0                          # 5. Interactive prompt (no echo)
```

> **`--pin` leaves the PIN in your shell history and in the process list.** Use `--pin-env` or `--pin-file` in production. A PIN never appears in any output or error message.
>
> When stdin is not a terminal, the prompt reads one line from stdin instead (the prompt text goes to stderr). **An empty PIN is never sent**: kmu exits with code 2, since an empty PIN would still count as a failed login on the HSM.

Roles

Logs in as Crypto Officer by default. Add `--crypto-user` for the Crypto User role.

Verifying credentials

```
kmu login --slot 0 --pin-env HSM_PIN
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer
```

Connects, logs in, prints the success line, and exits — no state is kept.

```
kmu logout
```

Output:

```
單次執行模式不持有 session：每個指令各自登入、執行完即結束，因此不需要登出。
```

`logout` is an explicit no-op — it prints its message and exits. kmu runs one command at a time and never holds a session, so there is nothing to log out of.

---

3. Inspection commands

List slots

```
kmu listslot        # `kmu listslots` is an accepted alias
```

Output:

```
Slot list :
Slot  Label  Model         Firmware  Software  Serial
0     10     LunaSA 7.9.1  7.9.3     7.9.1     1482011006884
1     1      LunaSA 7.9.1  7.9.3     7.9.1     1482011006875
2     6      LunaSA 7.9.1  7.9.3     7.9.1     1482011006880
```

List objects

```
kmu list --slot 0 --pin-env HSM_PIN
kmu list --slot 0 --pin-env HSM_PIN --limit 20     # first 20 only
kmu list --slot 0 --pin-env HSM_PIN --json         # for scripts
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

handle=1417  Class=Secret Key   Label=mykek
handle=1925  Class=Private Key  Label=myrsa
handle=1365  Class=Public Key   Label=myrsa
handle=1866  Class=Secret Key   Label=mykey

 4 objects found
```

Show one object's attributes

```
kmu getattribute --slot 0 --pin-env HSM_PIN --handle 1866
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

The list of attributes for handle 1866 are :

Class=Secret Key
Token=true
Private=true
Modifiable=true
Label=mykey
ObjectSize=264 bytes
KeyType=aes (0x0000001F)
id=
StartDate=
EndDate=
Local=true
Derive=true
Sensitive=true
AlwaysSensitive=true
Extractable=true
NeverExtractable=false
```

```
Encrypt=true
Decrypt=true
Sign=true
Verify=true
Derive=true
…(remaining rows are Luna-specific attributes)
```

Covers class, key type, label, capability flags, fingerprints and vendor attributes.

Show supported mechanisms

```
kmu getcapabilities --slot 0
```

Output:

```
Supported key generation mecanisms :
-> aes              Minimum key size : 16       Maximum key size : 32
-> sm4              Key size : 128
-> des              Key size : 64
-> des2             Key size : 128
-> des3             Key size : 192
-> genericsecret        Minimum key size : 64       Maximum key size : 4096
-> hmac             Minimum key size : 64       Maximum key size : 4096
-> rsa-pkcs         Minimum key size : 256      Maximum key size : 8192
…(other key types follow the same format)
```

---

4. Generating keys

Symmetric keys

```
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type aes --key-size 32 --label mykey
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Key successfully generated, handle is : 1311, label is : mykey
```

`--key-type`	Allowed `--key-size` (bytes)
`aes`, `sm4`	16, 24, 32
`des`	8
`des2`	16
`des3`	24
`genericsecret`, `hmac`	1–512

Asymmetric key pairs

```
# RSA (--mech accepts pkcs, prime, aux). --key-size is the modulus length in
# BITS, not bytes -- 256 silently gives you a 256-bit RSA key
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type rsa --key-size 2048 --label myrsa

# A longer RSA key
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type rsa --key-size 4096 --label myrsa40
96

# EC family: pick the curve with --curve
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type ecdsa --curve prime256v1 --label my
ec
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type ecdsa --curve secp384r1 --label myp
384
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type ecdsa --curve secp256k1 --label myk
1
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type eddsa --curve ed25519 --label myed
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type eddsa --curve ed448 --label myed448
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type montgomery --curve x25519 --label m
yx

# Post-quantum: pick the parameter set (short forms 65 / 768 also work;
# non-alphanumeric characters are ignored)
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type ml-dsa --parameter-set ml-dsa-65 --
label mypqc
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type ml-kem --parameter-set ml-kem-768 -
-label mykem

# DSA/DH need domain parameters
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type dsa --label mydsa \
    --prime <p in hex> --subprime <q> --base <g>
```

```
# HSS/LMS: --lms-type and --lmots-type are comma-separated, one per level
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type hss --label myhss \
    --hss-levels 2 --lms-type 5,5 --lmots-type 4,4
```

A key pair prints both handles:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Key pair successfully generated, private key handle : 1405, public key handle : 639
```

Supported types: `rsa`, `dsa`, `dh`, `dh-x9.42`, `ecdsa`, `eddsa`, `montgomery`, `sm2`, `ml-dsa`, `ml-kem`, `hss/lms`.

Use `--label-public` and `--label-private` to name the halves separately; a single `--label` applies to both.

Attribute flags

These default to **`true` when not specified**:

`--token`, `--private-object`, `--sensitive`, `--extractable`, `--modifiable`, `--encrypt`, `--decrypt`, `--sign`, `--verify`, `--wrap`, `--unwrap`, `--derive`

Values are lowercase `true`/`false` only (`1`/`0` is an argument error):

```
kmu generatekey ... --sign false --wrap true
```

A flag only applies where the attribute makes sense for that object. For a key pair, the public key carries no `CKA_SENSITIVE` or `CKA_EXTRACTABLE`, and `--sign`, `--decrypt` and `--unwrap` affect only the private key; `--private-object` applies to both halves, so by default the public key is `Private=true` as well.

Clear key components

Builds a key from the XOR of N components. Components are read without echo, and the KCV of the assembled key is shown at the end using the `pci` method (`sm4`, `genericsecret` and `hmac` have no matching KCV method, so none is shown). Each component must be exactly `--key-size` bytes; this mode cannot be combined with `--json`:

```
kmu generatekey --slot 0 --pin-env HSM_PIN --key-type aes --key-size 32 \
    --label combined --clear-components 3
```

---

5. Object management

Create a data object

```
kmu createdo --slot 0 --pin-env HSM_PIN --label mydata --application app --value 41424344
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Data Object successfully generated, handle value is 1902,
```

Modify attributes

Unspecified attributes are left alone.

```
kmu setattribute --slot 0 --pin-env HSM_PIN --handle 1866 --sign false --label mykey
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Attempt to update attributes on object 1866 ...
Updating attribute CKA_SIGN ... Success. Value = false

2 attributes updated
```

Text attributes such as `--label` print no line of their own when they succeed, but they are counted in the total.

> **Not every attribute can be changed after creation:**
>
> - `--extractable` can only go from `true` to `false`; setting it back to `true` fails.
> - `--private-object` can only be chosen when the object is created (`generatekey`, `import`, …). Changing it afterwards with `setattribute` always fails, even when setting the same value (tested on Luna firmware 7.9.3).
> - `--token` and `--sensitive` are not among the attributes `setattribute` sends (same as upstream); specifying them **has no effect**.
> - `AlwaysSensitive` and `NeverExtractable` are maintained by the HSM; no command can change them.
>
> Each attribute is sent and reported separately; `--json` lists them in `results`.

Read and write a single attribute

`CKA_ID` and `CKA_VALUE` are read and written as binary, everything else as text.

```
kmu readattribute  --slot 0 --pin-env HSM_PIN --handle 1866 --attribute id --output-file
id.bin
kmu writeattribute --slot 0 --pin-env HSM_PIN --handle 1866 --attribute id --input-file i
d.bin
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Successfull: 4 bytes written in file : id.bin
```

Digest and KCV

```
kmu computekcv --slot 0 --pin-env HSM_PIN --handle 1866 --method pci
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

KCV of the key 1866 is equal to : 620853
```

`digestkey` hashes the key itself and is used the same way:

```
kmu digestkey --slot 0 --pin-env HSM_PIN --handle 1866 --hash sha256
```

It is governed by partition policy and always fails when that policy is off (see section 9).

Delete

The object is shown and confirmation is required by default; non-interactive environments (including `--json`) must pass `--yes`.

```
kmu delete --slot 0 --pin-env HSM_PIN --handle 1902
kmu delete --slot 0 --pin-env HSM_PIN --handle 1902 --yes    # skip the prompt
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Object with handle 1902 deleted
```

The four KCV methods

`--method`	AES	DES family	generic secret
`pkcs11`	ECB-encrypt an all-zero block	ECB-encrypt an all-zero block	—
`pci`	AES-CMAC (all-zero input)	same as `pkcs11`	—
`gp`	AES-CMAC over all-`0x01` input	same as `pkcs11`	—
`hmac-sha-256`	—	—	HMAC-SHA-256

All take the **first 3 bytes** of the result. For the DES family the three methods give the same value; SM4 keys support none of them.

> **The same key yields different values under different methods.** When comparing KCVs against another tool, confirm both sides use the same one.

`computekcv` temporarily modifies the object. If the key lacks the required `CKA_SIGN` or `CKA_ENCRYPT`, kmu enables it, computes, and restores it — without that the KCV cannot be computed. The restore also runs if the operation fails partway, and `--json` reports which attribute was touched.

---

6. Cryptographic operations

Encrypt and decrypt

```
kmu encrypt --slot 0 --pin-env HSM_PIN --handle 1866 --algo aes_cbc_pad \
    --input-file plain.bin --output-file cipher.bin
kmu decrypt --slot 0 --pin-env HSM_PIN --handle 1866 --algo aes_cbc_pad \
    --input-file cipher.bin --output-file plain.bin
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Encrypt successfull: 16 bytes written in file : cipher.bin
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Decrypt successfull : 5 bytes written in file : plain2.bin
```

Algorithms (15): `aes_ecb`, `aes_cbc`, `aes_cbc_pad`, `aes_cbc_pad_ipsec`, `aes_cfb8`, `aes_cfb128`, `aes_ofb`, `aes_kw`, `aes_kwp`, `aes_gcm`, `rsa_pkcs`, `rsa_oaep_sha256`, `rsa_oaep_sha384`, `rsa_oaep_sha512`, `rsa_oaep`

```
# --format governs BOTH input and output: bin (raw) or text (uppercase hex)
kmu encrypt ... --format text

# Modes needing an IV have a default; --iv overrides it
kmu encrypt ... --algo aes_cbc --iv 000102030405060708090A0B0C0D0E0F

# GCM extras
kmu encrypt ... --algo aes_gcm --aad 41424344 --tag-bits 128

# rsa_oaep does not encode the hash in its name — use --hash
# (sha1 / sha256 / sha384 / sha512)
kmu encrypt ... --algo rsa_oaep --hash sha256
```

Key derivation

```
kmu derive --slot 0 --pin-env HSM_PIN --handle 1866 --mech sha256 \
    --key-type aes --key-size 32 --label derived
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Key successfully derived: handle is : 1271, label is : derived
```

Mechanisms (12): `sha1`, `sha224`, `sha256`, `sha384`, `sha512`, `sha3-224`, `sha3-256`, `sha3-384`, `sha3-512`, `luna-kdf`, `luna-nist-kdf`, `aes-encrypt-ecb`

```
# Luna KDF needs extra parameters
kmu derive ... --mech luna-nist-kdf --kdf-type aes-cmac --kdf-scheme scheme1 \
    --kdf-counter 1 --kdf-label 41424344 --kdf-context 50515253
```

```
# Derive by encrypting data
# --kdf-data must be long enough to derive a key of --key-size: deriving a
# 32-byte AES key needs 32 bytes of data.
kmu derive ... --mech aes-encrypt-ecb \
    --kdf-data 000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
```

--kdf-type: tdes-cmac, aes-cmac, hmac-sha1...hmac-sha512 --kdf-scheme: scheme1...scheme4, scp03, hid

---

7. Export and import

Public keys

```
kmu export --slot 0 --pin-env HSM_PIN --handle 1365 --format pkcs8 --output-file pub.pem
kmu import --slot 0 --pin-env HSM_PIN --key-class public --key-type ecdsa \
    --label mypub --input-file pub.pem --format pkcs8
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Export public key : Key (handle=1365) successfully exported.
451 bytes written in file : pub.pem
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Public key successfully imported: handle is : 490, label is : mypub
```

Formats: `bin` (SubjectPublicKeyInfo DER), `text` (the same as hex), `pkcs8` (`-----BEGIN PUBLIC KEY-----` PEM).

Supported types: RSA, DSA, DH/X9.42-DH, ECDSA, SM2, Ed25519, Ed448, X25519, **X448**, ML-DSA, ML-KEM, HSS.

`export` picks its path from the object's class — there is **no `--key-class`**.

Symmetric and private keys (wrapped under another key)

```
kmu export --slot 0 --pin-env HSM_PIN --handle 1866 --format bin \
    --algo aes_kwp --key 1417 --output-file key.bin
kmu import --slot 0 --pin-env HSM_PIN --key-class secret --key-type aes \
    --label mykey --input-file key.bin --format bin --algo aes_kwp --key 1417
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Export : Key (handle=1866) successfully wrapped with key (handle=1417).
40 bytes written in file : key.bin
```

Wrapping algorithms (12): the 15 encryption algorithms minus `aes_cfb8`, `aes_cfb128` and `aes_ofb` — stream modes cannot be used for wrapping.

> **Private key export is governed by partition policy.** If `Allow private key wrapping` is `0` you get `CKR_KEY_NOT_WRAPPABLE`. Check with `partition showpolicies` in `lunacm`.

Password-protected private keys (PKCS#8 PBES2)

```
kmu export --slot 2 --pin-env HSM_PIN --handle 1925 --format pkcs8 \
    --algo pbkdf2_aes256_cbc --prf hmac-sha1 --key-password 'secret' --output-file prv.pe
m
```

```
kmu import --slot 2 --pin-env HSM_PIN --key-class private --key-type rsa \
    --label myprv --input-file prv.pem --format pkcs8 --key-password 'secret'
```

Algorithms: `pbkdf2_aes128_cbc`, `pbkdf2_aes192_cbc`, `pbkdf2_aes256_cbc`. `--salt` and `--iv` default to HSM-generated random values; `--iteration` defaults to 10000.

On import, salt / iteration count / PRF / IV are **all read from the file** — those options do not exist on the import side.

The output interoperates with OpenSSL:

```
openssl pkey -in prv.pem -passin pass:secret -noout -text
```

> **Password case is significant here**: `Passw0rd` and `passw0rd` are two different passwords.

TR-31 key blocks

```
kmu export --slot 0 --pin-env HSM_PIN --handle 1866 --format tr31 --key 1548 \
    --tr31-usage K0 --tr31-mode B --tr31-export E --output-file block.txt
kmu import --slot 0 --pin-env HSM_PIN --key-class secret --key-type aes \
    --label imported --input-file block.txt --format tr31 --key 1548
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Export : Key (handle=1866) successfully wrapped with key (handle=1548).
144 bytes written in file : block.txt
```

Version `D` (AES Key Derivation Binding) and AES keys only. `--key` is the handle of the KBPK (Key Block Protection Key).

> Producing a TR-31 block requires the key's plaintext to pass briefly through host memory (the MAC covers the unencrypted payload and must be computed before encryption). It is wiped as soon as the block has been produced.

Clear-component import

```
kmu import --slot 0 --pin-env HSM_PIN --key-class secret --key-type aes \
    --key-size 32 --clear-components 3 --label combined
```

`aes`, `des`, `des2` and `des3` only; 2–16 components. Each is entered without echo, its KCV is displayed, and you confirm before continuing.

File encoding conversion

```
kmu convert --input-file a.bin --output-file a.txt --in-form bin --out-form text
kmu convert --input-file a.txt --output-file a.bin --in-form text --out-form bin
```

Output:

```

convert :
26 bytes written in file : a.txt
```

No HSM connection. `--in-form` and `--out-form` accept only `bin` and `text`, and must differ (`txt` is an alias for `text`, so the two count as the same).

---

8. Negotiating a key with a payShield TMD

```
kmu remotemzmk --slot 0 --pin-env HSM_PIN --key-type aes --key-size 32 \
    --label mzmk --input-file tmd_pub.txt
```

Output:

```
Authentication with slot [0] : 10 ... Success -> Connected as Crypto Officer

Key successfully derived: handle is : 1601, label is : mzmk
MZMK Key check value : 62727F

Successfull: the file MZMKdata_2026_09_16_08_33.csv has been generated. You can upload in
 TMD.
```

One ECDH exchange; both ends independently arrive at the same MZMK:

1. Read the TMD's EC public key (SubjectPublicKeyInfo as hex text)
2. Generate a session key pair on the same curve
3. Derive the MZMK with `CKM_ECDH1_DERIVE` + `CKD_SHA256_KDF` (X9.63)
4. Write `MZMKdata_<year>_<month>_<day>_<hour>_<minute>.csv` **into the input file's directory**

The key never leaves either end — the CSV carries only a public key and a KCV, neither of which is secret. Upload the CSV to the TMD and compare KCVs to confirm the exchange succeeded.

`--key-type` accepts `aes` (16/24/32 bytes) and `des` (24 only, created as DES3).

---

9. Troubleshooting

`CKR_KEY_NOT_WRAPPABLE`

Partition policy forbids exporting that class of key. Check with:

```
printf 'slot set -slot 0\npartition showpolicies\nexit\n' | lunacm | grep "private key wrapping"
```

`Allow private key wrapping : 0` means private key export is disabled on that partition (the `Enable private key wrapping` line that also matches is the HSM's capability, not the partition's policy). **Query one slot at a time** — chaining several slots through one pipeline stops `slot set` from taking effect and you will read the same partition's policies every time.

`CKR_OPERATION_NOT_ALLOWED` from `digestkey`

Partition policy `Allow digest key` is `0`. This is unrelated to the key's `CKA_SENSITIVE`.

KCV does not match another tool

Confirm both sides use the same method. `pkcs11` ECB-encrypts an all-zero block; `pci` is AES-CMAC for AES keys. The same key gives different values under each, and both are correct.

`CKR_ECC_UNKNOWN_CURVE`

kmu recognised the curve name, but this HSM does not support that curve. Run `kmu getcapabilities --slot N` to see what the slot actually supports. (A name kmu does not recognise is rejected as an unknown curve before anything reaches the HSM; common names such as `prime256v1`/`secp256r1`, `secp384r1`, `secp256k1`, `ed25519`, `ed448`, `x25519`, `x448` and `sm2` are all known.)

OpenSSL cannot read a PKCS#8 file

First check the command. **`openssl pkcs8 -in enc.pem -nocrypt` does not decrypt** — without `-topk8`, `-nocrypt` means "the input is unencrypted". Use:

```
openssl pkey -in enc.pem -passin pass:yourpass -noout -text
```

If that still fails and the password contains uppercase letters, try the all-lowercase form — some tools lowercase the password before deriving the key, and files they produce only open with the lowercased password.

Entering components and passwords in a pipeline

When stdin is not a terminal, clear components, PINs and PKCS#8 passwords are all read as a plain line, so scripting works:

```
printf 'y\n%s\ny\n%s\ny\n' "$C1" "$C2" | kmu import ... --clear-components 2
```

In scheduled jobs stdin is often empty (for example `</dev/null`); the PIN read is then empty and kmu exits with "PIN 是空的" instead of logging in. Prefer `--pin-env` or `--pin-file` for the PIN anyway.

PKCS#11 library not found

Check, in order:

1. `/etc/Chrystoki.conf` (Windows: `C:\Program Files\SafeNet\LunaClient\crystoki.ini`) exists, is readable, and its `Chrystoki2` section has `LibUNIX64` (Windows: `LibNT64` or `LibNT`). Only 64-bit keys are honoured — `LibUNIX`/`LibNT32` are the 32-bit builds and are ignored.
2. If the config lives elsewhere, `ChrystokiConfigurationPath` must name the **directory** holding it.
3. Otherwise give the path explicitly with `--library <path>`.

If `ChrystokiConfigurationPath` **is** set but no config file is there, both platforms fail immediately. When the config file is readable but lacks the keys above, or the variable is unset and the default config file is missing:

- Linux falls back to `dlopen("libCryptoki2_64.so")`; if that file is not on the loader's search path, the error is deferred to load time.
- Windows has no fallback and fails immediately.

Configuration errors exit with code 2.