

Package ‘KeyNubLicDongle’

October 8, 2026

Type Package

Title Client for the 'KeyNub' USB License Dongle

Version 1.1.1

Description Verifies that a 'KeyNub' USB license dongle is genuine, reads and writes the license records it holds, reads and increments its monotonic counters, and encrypts data so that only a dongle can decrypt it. The work is done by the vendor's native library, which the package loads at run time; the package installs without it and reports where it looked. Runs on Windows, Linux and macOS.

License Apache License (== 2.0)

URL <https://www.keynub.com/developers/r/>,
<https://github.com/AB-KeyNub/KeyNub-SDK>

BugReports <https://github.com/AB-KeyNub/KeyNub-SDK/issues>

SystemRequirements the KeyNub License Dongle native library
(keynub_licdongle) for the platform, see
<https://github.com/AB-KeyNub/KeyNub-SDK/blob/master/NATIVES.md>

Encoding UTF-8

Depends R (>= 4.0)

NeedsCompilation yes

Config/roxygen2/version 8.1.0

Author Andreas Breitschopp [aut, cre],
KeyNub [cph]

Maintainer Andreas Breitschopp <info@keynub.com>

Repository CRAN

Date/Publication 2026-10-08 10:10:06 UTC

Contents

licd_app_encrypt	2
licd_close	3

licd_context	4
licd_counter_read	4
licd_enumerate	5
licd_error	6
licd_error_detail	6
licd_info	7
licd_is_open	8
licd_library	8
licd_open	9
licd_record_list	10
licd_record_read	10
licd_serial	11
licd_session_open	12
licd_set_trust_root	13
licd_status_codes	13
licd_strerror	14
licd_verify_genuine	14
licd_version	15
licd_write_auth	16

Index	17
--------------	-----------

licd_app_encrypt	<i>Encrypt data that only a dongle can decrypt</i>
------------------	--

Description

The pair to build a licence check on. licd_app_encrypt() seals data so that only a dongle can open it; licd_app_decrypt() opens it again. Put something the program genuinely needs through this, such as the parameters of your model, and ship only the sealed form: then removing the check removes the data. Bulk encryption runs on the host; the dongle wraps only a small key.

Usage

```
licd_app_encrypt(dongle, data, scope = c("device", "developer"))

licd_app_decrypt(dongle, packed)
```

Arguments

dongle	A licd_dongle with an open session.
data	The plaintext: a raw vector, or a single string for its bytes.
scope	"device" or "developer".
packed	The sealed data from licd_app_encrypt(), a raw vector.

Details

scope = "developer" lets any dongle you have issued decrypt the data, so one sealed file serves every customer; scope = "device" locks it to the one dongle that sealed it.

Value

A raw vector: the sealed data, or the recovered plaintext.

Examples

```
# Needs the native library and an attached dongle; does nothing without them.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
dongle <- if (!is.null(ctx)) tryCatch(licd_open(ctx), licd_error = function(e) NULL)
if (!is.null(dongle)) {
  sealed <- licd_with_session(dongle, licd_app_encrypt(dongle, "the data", "developer"))
  file <- tempfile(fileext = ".sealed")
  saveRDS(sealed, file)
  # In the shipped program:
  plain <- licd_with_session(dongle, licd_app_decrypt(dongle, readRDS(file)))
  print(rawToChar(plain))
  unlink(file)
}
if (!is.null(ctx)) licd_close(ctx) # closes the dongle as well
```

licd_close

Close a context or a dongle

Description

Releases the handle. Safe to call more than once. Closing a context closes every dongle still open on it. An object that is not closed is released when the garbage collector reclaims it, so a forgotten handle leaks nothing, but closing explicitly frees the dongle for other programs at once.

Usage

```
licd_close(x)
```

Arguments

x A licd_context from [licd_context\(\)](#) or a licd_dongle from [licd_open\(\)](#).

Value

NULL, invisibly.

Examples

```
# Does nothing where the native library is not installed.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
if (!is.null(ctx)) {
  dongle <- tryCatch(licd_open(ctx), licd_error = function(e) NULL)
  if (!is.null(dongle)) licd_close(dongle)
  licd_close(ctx)
}
```

licd_context	<i>Create a library context</i>
--------------	---------------------------------

Description

A context owns the connection to the operating system's USB layer and the dongles opened on it. One per program is usual. Close it with [licd_close\(\)](#) when done; a context the garbage collector reclaims is closed as well, together with any dongle still open on it.

Usage

```
licd_context()
```

Value

A `licd_context` object.

Examples

```
# Does nothing where the native library is not installed.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
if (!is.null(ctx)) {
  print(licd_enumerate(ctx))
  licd_close(ctx)
}
```

licd_counter_read	<i>Monotonic counters</i>
-------------------	---------------------------

Description

The dongle holds hardware counters that only ever go up. Reading needs a session; incrementing needs the write role and is irreversible.

Usage

```
licd_counter_read(dongle, id)

licd_counter_increment(dongle, id)
```

Arguments

<code>dongle</code>	A <code>licd_dongle</code> with an open session.
<code>id</code>	The counter, an integer from 0 upwards; the dongle reports <code>licd_status_codes[["range"]]</code> for one it does not have.

Value

The counter's value (after the increment, for `licd_counter_increment()`) as a number. Counters are unsigned 32-bit, so the value is returned as a double.

Examples

```
# Needs the native library and an attached dongle; does nothing without them.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
dongle <- if (!is.null(ctx)) tryCatch(licd_open(ctx), licd_error = function(e) NULL)
if (!is.null(dongle)) {
  print(licd_with_session(dongle, licd_counter_read(dongle, 0)))
}
if (!is.null(ctx)) licd_close(ctx) # closes the dongle as well
```

<code>licd_enumerate</code>	<i>List the attached dongles</i>
-----------------------------	----------------------------------

Description

Enumerates without opening anything.

Usage

```
licd_enumerate(ctx)
```

Arguments

<code>ctx</code>	A <code>licd_context</code> .
------------------	-------------------------------

Value

A data frame with one row per dongle and the columns `serial` (hex), `path` (the operating system's device path, accepted by [licd_open_path\(\)](#)), `vendor_id` and `product_id` (USB ids). Zero rows when none is attached.

Examples

```
# Does nothing where the native library is not installed.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
if (!is.null(ctx)) {
  dongles <- licd_enumerate(ctx)
  print(nrow(dongles))
  licd_close(ctx)
}
```

licd_error	<i>Errors signalled by this package</i>
------------	---

Description

Every failure the native library reports is signalled as a condition of class `licd_error` (and `error`), with these fields:

Details

- `status`: the code, one of [licd_status_codes](#);
- `operation`: the SDK function that failed, such as `"licd_open"`;
- `detail`: the library's diagnostic text for this failure, or `""`. Log it; do not parse it.

The failures a program is likely to branch on carry a more specific class in front of `licd_error`:

`licd_no_device`, `licd_not_genuine`, `licd_certificate_invalid`, `licd_session_expired`, `licd_not_found`, `licd_auth_required` and `licd_cancelled`.

Examples

```
# Does nothing where the native library is not installed.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
if (!is.null(ctx)) {
  dongle <- tryCatch(licd_open(ctx), licd_no_device = function(e) {
    message("no dongle attached")
    NULL
  })
  licd_close(ctx)
}
```

licd_error_detail	<i>Diagnostic detail of the most recent failure</i>
-------------------	---

Description

The native library's diagnostic text for the last call that failed on the current thread. The same text travels in the `detail` field of every [licd_error](#), so this is for code that inspects a failure after the fact.

Usage

```
licd_error_detail(ctx)
```

Arguments

`ctx` A `licd_context`.

Value

A string, `""` when the last call succeeded.

licd_info	<i>Plaintext device information</i>
-----------	-------------------------------------

Description

Reads what the dongle reports without a session.

Usage

```
licd_info(dongle)
```

Arguments

dongle A licd_dongle.

Value

A list: protocol_version (two integers), firmware_version (three integers), se_ready (the secure element responded), provisioned (factory provisioning complete), data_capacity and data_free (bytes), watchdog_reboot, isolated and writeauth_rotated (logicals).

watchdog_reboot means the dongle's *previous* boot ended in a watchdog timeout: the firmware hung and reset itself. It is the only trace a field hang leaves behind and a power cycle clears it, so it is worth logging.

writeauth_rotated is FALSE while the dongle still accepts the write key it left the factory with. That key is public, so a dongle in that state takes writes from anyone holding it; rotate on receipt with [licd_write_auth_rotate\(\)](#).

Examples

```
# Needs the native library and an attached dongle; does nothing without them.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
dongle <- if (!is.null(ctx)) tryCatch(licd_open(ctx), licd_error = function(e) NULL)
if (!is.null(dongle)) {
  info <- licd_info(dongle)
  print(info$firmware_version)
}
if (!is.null(ctx)) licd_close(ctx) # closes the dongle as well
```

licd_is_open	<i>Is a context or dongle still open?</i>
--------------	---

Description

Is a context or dongle still open?

Usage

```
licd_is_open(x)
```

Arguments

x A licd_context or a licd_dongle.

Value

TRUE while the handle can be used, FALSE once it has been closed, explicitly or through its context.

Examples

```
# Does nothing where the native library is not installed.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
if (!is.null(ctx)) {
  print(licd_is_open(ctx))
  licd_close(ctx)
  print(licd_is_open(ctx))
}
```

licd_library	<i>The native library in use</i>
--------------	----------------------------------

Description

The package does its work through the KeyNub native library (keynub_licdongle.dll, libkeynub_licdongle.so or libkeynub_licdongle.dylib), which a process loads once, on the first call that needs it. licd_library(path) names the file to load, and must come before that first call. Without it the package takes, in this order, the KEYNUB_LICDONGLE_LIBRARY environment variable, the natives/<platform>/ folder of a clone of the SDK repository found from the working directory upwards, and finally the bare file name, which the operating system resolves along its usual search path (PATH, LD_LIBRARY_PATH, DYLD_LIBRARY_PATH).

Usage

```
licd_library(path = NULL)
```

Arguments

path A file path to the library, or NULL to ask.

Value

The path in use, or the one that would be used: the loaded library once there is one, otherwise the choice the next call would make. Invisibly when path is given.

Examples

```
# The file the package has loaded, or would load on its first call.
licd_library()
```

licd_open	<i>Open a dongle</i>
-----------	----------------------

Description

licd_open() opens the dongle with the given serial, or the first one found when serial is NULL. licd_open_path() opens the dongle at a device path from [licd_enumerate\(\)](#). Both signal a condition of class licd_no_device when there is no such dongle.

Usage

```
licd_open(ctx, serial = NULL)

licd_open_path(ctx, path)
```

Arguments

ctx A licd_context.
 serial The serial as hex, or NULL for the first dongle.
 path A device path from [licd_enumerate\(\)](#).

Value

A licd_dongle object. Close it with [licd_close\(\)](#).

Examples

```
# Needs the native library and an attached dongle; does nothing without them.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
dongle <- if (!is.null(ctx)) tryCatch(licd_open(ctx), licd_error = function(e) NULL)
if (!is.null(dongle)) {
  print(licd_serial(dongle))
  licd_close(dongle)
}
if (!is.null(ctx)) licd_close(ctx)
```

licd_record_list	<i>The records on the dongle</i>
------------------	----------------------------------

Description

The records on the dongle

Usage

```
licd_record_list(dongle)
```

Arguments

dongle A licd_dongle with an open session.

Value

A data frame with the columns name and size (bytes), one row per record; zero rows when the dongle holds none.

Examples

```
# Needs the native library and an attached dongle; does nothing without them.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
dongle <- if (!is.null(ctx)) tryCatch(licd_open(ctx), licd_error = function(e) NULL)
if (!is.null(dongle)) {
  print(licd_with_session(dongle, licd_record_list(dongle)))
}
if (!is.null(ctx)) licd_close(ctx) # closes the dongle as well
```

licd_record_read	<i>Read, write and erase records</i>
------------------	--------------------------------------

Description

Records are named blobs in the dongle's storage; a licence typically is one. Reading needs a session, writing and erasing need the write role ([licd_write_auth\(\)](#)). `licd_record_write()` replaces the record atomically. A record that does not exist signals `licd_not_found`.

Usage

```
licd_record_read(dongle, name, progress = NULL)

licd_record_write(dongle, name, data, progress = NULL)

licd_record_erase(dongle, name)

licd_record_erase_all(dongle)
```

Arguments

dongle	A licd_dongle with an open session.
name	The record name, a non-empty string.
progress	NULL, or a function(done, total) called as the transfer proceeds, both arguments in bytes. Return FALSE to cancel, which signals licd_cancelled; an error inside the function cancels as well.
data	The bytes to store: a raw vector, or a single string for its bytes.

Details

licd_record_erase_all() is a separate function on purpose: to the C library a missing name means "erase every record", and an accidentally empty variable must not do that.

Value

licd_record_read(): the record as a raw vector (rawToChar() gives text back). The others: NULL, invisibly.

Examples

```
# Needs the native library and an attached dongle; does nothing without them.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
dongle <- if (!is.null(ctx)) tryCatch(licd_open(ctx), licd_error = function(e) NULL)
if (!is.null(dongle)) {
  licd_with_session(dongle, {
    records <- licd_record_list(dongle)
    if (nrow(records) > 0) print(licd_record_read(dongle, records$name[1]))
  })
}
if (!is.null(ctx)) licd_close(ctx) # closes the dongle as well
```

licd_serial	<i>The dongle's serial</i>
-------------	----------------------------

Description

The dongle's serial

Usage

```
licd_serial(dongle)
```

Arguments

dongle	A licd_dongle.
--------	----------------

Value

The serial as a hex string.

licd_session_open	<i>Open and close the encrypted session</i>
-------------------	---

Description

Records, counters and app-data encryption need a session: an encrypted, authenticated channel to the dongle (P-256 ECDH, HKDF-SHA256, AES-256-GCM). `licd_with_session()` opens one, evaluates `expr` and closes the session afterwards whatever happens.

Usage

```
licd_session_open(dongle)

licd_session_close(dongle)

licd_with_session(dongle, expr)
```

Arguments

<code>dongle</code>	A <code>licd_dongle</code> .
<code>expr</code>	An expression to evaluate while the session is open.

Value

`licd_session_open()` and `licd_session_close()`: `NULL`, invisibly. `licd_with_session()`: the value of `expr`.

Examples

```
# Needs the native library and an attached dongle; does nothing without them.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
dongle <- if (!is.null(ctx)) tryCatch(licd_open(ctx), licd_error = function(e) NULL)
if (!is.null(dongle)) {
  licd_session_open(dongle)
  print(licd_record_list(dongle))
  licd_session_close(dongle)
  # The same, closing the session whatever happens:
  print(licd_with_session(dongle, licd_record_list(dongle)))
}
if (!is.null(ctx)) licd_close(ctx) # closes the dongle as well
```

licd_set_trust_root	<i>Override the trust root</i>
---------------------	--------------------------------

Description

Replaces the CA root certificate that `licd_verify_genuine()` checks the dongle's certificate chain against. Applications do not need this: a release build of the library embeds the KeyNub production root. It exists for dongles provisioned against a different CA, and for vendor tooling.

Usage

```
licd_set_trust_root(ctx, der)
```

Arguments

ctx	A <code>licd_context</code> .
der	The root certificate, DER-encoded, as a raw vector.

Value

NULL, invisibly.

licd_status_codes	<i>Status codes of the SDK</i>
-------------------	--------------------------------

Description

A named integer vector of the status codes the native library reports; ok is zero and every failure is negative. A failed call signals a condition of class `licd_error` whose status field holds one of these.

Usage

```
licd_status_codes
```

Format

A named integer vector with 21 entries.

Value

A named integer vector: each name is a status (such as `no_device`) and each value the code the native library returns for it, 0 for ok and negative for a failure. Compare it with the status field of a `licd_error`.

See Also

[licd_strerror\(\)](#), [licd_error](#)

Examples

```
licd_status_codes[["no_device"]]
names(licd_status_codes)[licd_status_codes == -14L]
```

licd_strerror	<i>Text for a status code</i>
---------------	-------------------------------

Description

Text for a status code

Usage

```
licd_strerror(status)
```

Arguments

status One or more status codes, see [licd_status_codes](#).

Value

A character vector with the native library’s short text for each code.

Examples

```
# NULL where the native library is not installed.
tryCatch(licd_strerror(licd_status_codes[["no_device"]]), error = function(e) NULL)
```

licd_verify_genuine	<i>Prove that a dongle is genuine</i>
---------------------	---------------------------------------

Description

Verifies the dongle’s certificate chain against the trusted root and runs a live challenge-response against the key inside the dongle. `licd_verify_genuine()` signals a condition of class `licd_not_genuine` (or `licd_certificate_invalid`) when the proof fails; `licd_is_genuine()` is the non-signalling form for a gate and **fails closed**: a missing dongle, an I/O error and an invalid certificate all give FALSE.

Usage

```
licd_verify_genuine(dongle)
```

```
licd_is_genuine(dongle)
```

Arguments

`dongle` A `licd_dongle`.

Details

Read the SDK's integration security guide before building a check on the answer. `if (!licd_is_genuine(dongle)) stop()` is one line to delete from a program that ships as source; what cannot be deleted is data the program needs and only the dongle can decrypt, see [licd_app_encrypt\(\)](#).

Value

`licd_verify_genuine()`: a list with `genuine` (TRUE), `serial` (from the verified certificate) and `provisioned_date` ("YYYY-MM-DD", or ""). `licd_is_genuine()`: TRUE or FALSE.

Examples

```
# Needs the native library and an attached dongle; does nothing without them.
ctx <- tryCatch(licd_context(), error = function(e) NULL)
dongle <- if (!is.null(ctx)) tryCatch(licd_open(ctx), licd_error = function(e) NULL)
if (!is.null(dongle)) {
  print(licd_is_genuine(dongle))
  result <- tryCatch(licd_verify_genuine(dongle), licd_error = function(e) e)
  print(if (inherits(result, "licd_error")) conditionMessage(result) else result$serial)
}
if (!is.null(ctx)) licd_close(ctx)    # closes the dongle as well
```

<code>licd_version</code>	<i>Version of the native library</i>
---------------------------	--------------------------------------

Description

Version of the native library

Usage

```
licd_version()
```

Value

An integer vector of length three: major, minor and patch version of the loaded library, which is the SDK version it was built from.

Examples

```
# NULL where the native library is not installed.
tryCatch(licd_version(), error = function(e) NULL)
```

licd_write_auth	<i>Elevate to the write role</i>
-----------------	----------------------------------

Description

licd_write_auth() unlocks writing, erasing and counter increments for the rest of the session with the dongle's write key, a P-256 private key in PKCS#8 DER. This belongs in your licence-issuing tooling; never ship that key in the application your users run. A key the dongle does not accept signals licd_not_genuine.

Usage

```
licd_write_auth(dongle, key)

licd_write_auth_rotate(dongle, key)
```

Arguments

dongle	A licd_dongle with an open session.
key	The key as a raw vector, for example readBin(path, "raw", file.size(path)).

Details

licd_write_auth_rotate() replaces the dongle's write key with key, a key you hold. It needs the write role, so call licd_write_auth() with the current key first. That session keeps the write role; from the next session on only the new key elevates, and the old one no longer works on this dongle. Do this once per dongle, when it arrives: the factory key is public.

Value

NULL, invisibly.

Examples

```
# Needs your write key, and replaces a record on the dongle.
## Not run:
key <- readBin("my-write-key.der", "raw", file.size("my-write-key.der"))
licd_with_session(dongle, {
  licd_write_auth(dongle, key)
  licd_record_write(dongle, "license", "expires 2027-12-31")
})

## End(Not run)
```

Index

licd_app_decrypt (licd_app_encrypt), [2](#)
licd_app_encrypt, [2](#)
licd_app_encrypt(), [15](#)
licd_close, [3](#)
licd_close(), [4](#), [9](#)
licd_context, [4](#)
licd_context(), [3](#)
licd_counter_increment
 (licd_counter_read), [4](#)
licd_counter_read, [4](#)
licd_enumerate, [5](#)
licd_enumerate(), [9](#)
licd_error, [6](#), [6](#), [13](#), [14](#)
licd_error_detail, [6](#)
licd_info, [7](#)
licd_is_genuine (licd_verify_genuine),
 [14](#)
licd_is_open, [8](#)
licd_library, [8](#)
licd_open, [9](#)
licd_open(), [3](#)
licd_open_path (licd_open), [9](#)
licd_open_path(), [5](#)
licd_record_erase (licd_record_read), [10](#)
licd_record_erase_all
 (licd_record_read), [10](#)
licd_record_list, [10](#)
licd_record_read, [10](#)
licd_record_write (licd_record_read), [10](#)
licd_serial, [11](#)
licd_session_close (licd_session_open),
 [12](#)
licd_session_open, [12](#)
licd_set_trust_root, [13](#)
licd_status_codes, [6](#), [13](#), [14](#)
licd_strerror, [14](#)
licd_strerror(), [14](#)
licd_verify_genuine, [14](#)
licd_verify_genuine(), [13](#)
licd_version, [15](#)
licd_with_session (licd_session_open),
 [12](#)
licd_write_auth, [16](#)
licd_write_auth(), [10](#)
licd_write_auth_rotate
 (licd_write_auth), [16](#)
licd_write_auth_rotate(), [7](#)