

Package: noqtis (via r-universe)

October 31, 2026

Type Package

Title Post-Quantum Signing, Key Encapsulation and Sealed Folders

Version 0.1.2

Description A post-quantum cryptography toolkit for R, built on the dependency-free 'NOQTIS' C library. Generate key pairs, then sign and verify with lattice-based 'ML-DSA', hash-based 'SLH-DSA', 'Ed25519' or hybrid 'ML-DSA'+ 'Ed25519' signatures that need both halves to verify; establish shared secrets with 'ML-KEM' or 'HQC' key encapsulation; and hash with 'SHA-2', 'SHA-3' or 'SHAKE', with 'HMAC'. A whole directory can be sealed under one signature and later verified offline; a Merkle proof shows chosen files to someone without revealing the others, and signed statements rotate or revoke a sealing key. Results can be wrapped in 'DSSE' envelopes. Keys and nonces come from the operating system's random generator, not from R's.

License Apache License (== 2.0) | file LICENSE

Copyright see file inst/COPYRIGHTS

URL <https://github.com/rootcoder007/noqtis>

BugReports <https://github.com/rootcoder007/noqtis/issues>

Depends R (>= 4.1)

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation yes

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Repository <https://rootcoder007.r-universe.dev>

Date/Publication 2026-10-31 00:38:40 UTC

RemoteUrl <https://github.com/rootcoder007/noqtis>
RemoteRef v0.1.2
RemoteSha 383a20ec194e49b14fc7ac613871cd23b7c45133
RemoteSubdir bindings/r

Contents

nqalgs	2
nqcapid	3
nqdsse	4
nqencap	5
nqhash	6
nqidcard	7
nqkeygen	8
nqrand	9
nqreadkey	9
nqrotate	10
nqseal	11
nqsign	13
nqver	14
Index	15

nqalgs	<i>Algorithms and their sizes</i>
--------	-----------------------------------

Description

Lists every signature and key encapsulation algorithm the bundled library provides, with the sizes in bytes of its keys, signatures, ciphertexts and shared secrets.

Usage

nqalgs()

Value

A data frame with one row per algorithm and the columns name, type ("sig" or "kem"), pk, sk, sig, ct and ss (sizes in bytes; NA where they do not apply) and id (the library's number for the algorithm).

Examples

nqalgs()

nqcapid	<i>Capsule ids, files and proofs</i>
---------	--------------------------------------

Description

nqcapid() gives a capsule's id: the SHA-256 of the bytes its seal signs, which nqseal() records for its inputs. nqfiles() lists the files a capsule's manifest holds. Neither checks anything: use nqcheck() for that.

Usage

```
nqcapid(dir)

nqfiles(dir)

nqprove(dir, paths)
```

Arguments

dir	Path of the capsule folder.
paths	The files to prove, as the manifest lists them (/ between folders).

Details

nqprove() writes a proof (noqtis-proof-1, JSON) that some files are in a format 2 capsule: the seal, the manifest's head and, per file, its entry and Merkle audit path (RFC 9162). The proof holds nothing of the capsule's other files. nqcheck(proof =) checks it, alone or against the files.

Value

nqcapid(): a raw vector of 32 bytes. nqfiles(): a data frame with the columns path, size and sha256 (64 hex digits), in manifest order. nqprove(): the proof, a raw vector.

Examples

```
d <- file.path(tempdir(), "proved")
dir.create(d)
writeLines("one", file.path(d, "a.txt"))
writeLines("two", file.path(d, "b.txt"))
k <- nqkeygen("ML-DSA-44")
nqseal(d, k$sk, k$pk, "ML-DSA-44", format = 2)
nqfiles(d)
p <- nqprove(d, "a.txt")
attr(nqcheck(NULL, k$pk, proof = p), "ok")
nqcapid(d)
unlink(d, recursive = TRUE)
```

nqdsse

*DSSE envelopes***Description**

nqdsse() wraps a payload in a DSSE envelope (the in-toto and SLSA signing wrapper), signed with any signature algorithm of [nqalgs\(\)](#) over the envelope's pre-authentication encoding, as canonical JSON with payload, payloadType and one signature whose keyid is the hex SHA-256 of pk. nqdsseverify() checks one.

Usage

```
nqdsse(sk, pk, payload, type, alg)
```

```
nqdsseverify(pk, env, alg)
```

Arguments

sk, pk	The key pair (nqdsse()), or the public key to check with (nqdsseverify()).
payload	The payload: a raw vector, or one string, taken as its UTF-8 bytes.
type	The payload type, 1 to 255 printable ASCII characters with no space, quote or backslash, such as "application/vnd.in-toto+json".
alg	Signature algorithm name.
env	The envelope: a raw vector, or one string.

Value

nqdsse(): the envelope, a raw vector. nqdsseverify(): a list with type and payload (a raw vector) when one of the envelope's signatures verifies under pk, NULL when none does; an error when env is not an envelope.

Examples

```
k <- nkeygen("ML-DSA-65")
e <- nqdsse(k$sk, k$pk, '{"a":1}', "application/json", "ML-DSA-65")
v <- nqdsseverify(k$pk, e, "ML-DSA-65")
rawToChar(v$payload)
```

nqencap

*Key encapsulation***Description**

nqencap() makes a fresh shared secret and its ciphertext for a public key (ML-KEM, FIPS 203, or HQC); nqdecap() recovers the shared secret from the ciphertext with the secret key.

Usage

```
nqencap(pk, alg, seed = NULL)
```

```
nqdecap(sk, ct, alg)
```

Arguments

pk	Public key, a raw vector (as from nqkeygen()).
alg	Key encapsulation algorithm name, one of the "kem" rows of nqalgs() .
seed	NULL (the default) for fresh randomness, or the standard's encapsulation seed as a raw vector (m, 32 bytes, for ML-KEM; m and salt for HQC), for test vectors.
sk	Secret key, a raw vector.
ct	Ciphertext, a raw vector.

Details

Decapsulation uses implicit rejection, as FIPS 203 and HQC require: a changed ciphertext gives an unrelated pseudorandom secret, not an error. Only a key or ciphertext of the wrong length is an error.

Value

nqencap(): a list with the raw vectors ct (ciphertext) and ss (shared secret). nqdecap(): the shared secret, a raw vector.

Examples

```
k <- nqkeygen("ML-KEM-768")
e <- nqencap(k$pk, "ML-KEM-768")
identical(nqdecap(k$sk, e$ct, "ML-KEM-768"), e$ss)
```

nqhash

*Hashes and HMAC***Description**

nqhash() computes a SHA-2 (FIPS 180-4), SHA-3 or SHAKE (FIPS 202) digest of a raw vector, a string or a file; nqhmac() computes an HMAC.

Usage

```
nqhash(x, alg = "sha256", file = FALSE, hex = FALSE, len = NULL)
```

```
nqhmac(key, x, alg = "sha256", hex = FALSE)
```

Arguments

x	A raw vector, or one string (hashed as its UTF-8 bytes), or, with file = TRUE, the path of a file.
alg	Hash name.
file	TRUE to hash the file at path x, read in chunks of 1 MiB.
hex	TRUE to return lowercase hexadecimal text instead of raw bytes.
len	For SHAKE only, the output length in bytes (default 32 for SHAKE128 and 64 for SHAKE256).
key	HMAC key, a raw vector or one string.

Details

Hash names: "sha224", "sha256", "sha384", "sha512", "sha512-224", "sha512-256", "sha3-224", "sha3-256", "sha3-384", "sha3-512", "shake128", "shake256" (case is ignored).

Value

The digest, a raw vector, or a string when hex = TRUE.

Examples

```
nqhash("abc", hex = TRUE)
nqhash(charToRaw("abc"), "sha3-256")
nqhash("abc", "shake128", len = 16)
f <- tempfile()
writeBin(charToRaw("abc"), f)
nqhash(f, file = TRUE, hex = TRUE)
unlink(f)
nqhmac("key", "The quick brown fox jumps over the lazy dog", hex = TRUE)
```

Description

An identity card (noqtis-idcard-1, canonical JSON) ties a signing key to who holds it: a name and, when given, an ORCID iD, an e-mail address, a web page and an affiliation, signed by the key itself under the card's own context, so that a card is never taken for a seal or a key statement. `nqidcard()` makes one; `nqidcheck()` checks one.

Usage

```
nqidcard(
    sk,
    pk,
    alg,
    name,
    orcid = NULL,
    email = NULL,
    url = NULL,
    affil = NULL,
    at = Sys.time()
)

nqidcheck(card, pk = NULL)
```

Arguments

<code>sk, pk</code>	The key pair the card is for; for <code>nqidcheck()</code> , <code>pk</code> is <code>NULL</code> or the public key the card must be for.
<code>alg</code>	Its signature algorithm.
<code>name</code>	Your name: 1 to 255 bytes of text, no control character, no space at either end.
<code>orcid</code>	<code>NULL</code> , or your ORCID iD ("0000-0002-1825-0097" or its <code>https://orcid.org/</code> address); its check digit must be right, and the card holds the bare iD.
<code>email</code>	<code>NULL</code> , or an e-mail address (<code>local@domain.tld</code> , at most 254 bytes).
<code>url</code>	<code>NULL</code> , or an <code>https://</code> web page (at most 1024 bytes).
<code>affil</code>	<code>NULL</code> , or your affiliation, as name.
<code>at</code>	When the card is made: a <code>POSIXct</code> time or seconds since 1970 (UTC); default <code>now</code> .
<code>card</code>	A card: a raw vector, or the path of a card file.

Details

A card is self-signed: it shows that whoever holds the key wrote it and that it is unchanged, not that the name is true. Give the key's SHA-256 (key_sha256) where people already know you, or let readers pin your public key with pk. The key must be post-quantum: Ed25519 alone is refused. `noqtis check --card` names the sealer of a capsule from its card.

Value

`nqidcard()`: the card, a raw vector (write it with `writeBin()`). `nqidcheck()`: a list with name, orcid, email, url, affil (NULL for a field the card has not), created, alg, pk and key_sha256 when the card verifies (and is for pk); NULL when its signature does not verify or it is for another key. Input that is not a card is an error.

Examples

```
k <- nqkeygen("ML-DSA-44")
card <- nqidcard(k$sk, k$pk, "ML-DSA-44", "Ada Lovelace",
  orcid = "0000-0002-1825-0097")
nqidcheck(card, k$pk)$orcid
```

nqkeygen

Generate a key pair

Description

Generates a signature key pair (ML-DSA, SLH-DSA, Ed25519, the hybrids ML-DSA-44/65/87+Ed25519) or a key encapsulation key pair (ML-KEM, HQC).

Usage

```
nqkeygen(alg, seed = NULL)
```

Arguments

alg	Algorithm name, one of <code>nqalgs()</code> \$name (case is ignored).
seed	NULL (the default) for fresh randomness from the operating system, or a raw vector holding the standard's seed for deterministic key generation: xi (32 bytes) for ML-DSA; SK.seed, SK.prf and PK.seed (3n bytes) for SLH-DSA; the 32-byte private key for Ed25519; xi and the Ed25519 key (64 bytes) for a hybrid; d and z (64 bytes) for ML-KEM; the HQC seed.

Value

A list with the raw vectors pk (public key) and sk (secret key) in the standard's encoding, and alg, the algorithm name.

See Also

[nqsign\(\)](#), [nqencap\(\)](#).

Examples

```
k <- nqkeygen("ML-DSA-44")
length(k$pk)
k2 <- nqkeygen("ML-KEM-512", seed = as.raw(0:63))
```

nqrand	<i>Random bytes from the operating system</i>
--------	---

Description

Bytes from the operating system's cryptographically secure generator, the same source key generation uses (see the Randomness section of [noqtis-package](#)). Independent of [set.seed\(\)](#).

Usage

```
nqrand(n)
```

Arguments

n Number of bytes.

Value

A raw vector of length n.

Examples

```
nqrand(16)
```

nqreadkey	<i>Read and write the command-line tool's key files</i>
-----------	---

Description

nqreadkey() reads a key file written by noqtis keygen (or by nqwritekey()), as strictly as the command-line tool reads it: the exact header line noqtis-key-1 <algorithm> <pub|sec>, then the key as padded base64 with LF or CRLF line ends and no other character, canonical padding, and exactly the algorithm's key size. nqwritekey() writes one.

Usage

```
nqreadkey(path)
```

```
nqwritekey(path, key, alg, kind = c("sec", "pub"), force = FALSE)
```

Arguments

path	The key file.
key	The key, a raw vector (\$pk or \$sk of nqkeygen()).
alg	The algorithm name, one of <code>nqalgs()</code> \$name (case is ignored).
kind	"pub" for a public key, "sec" for a secret key.
force	TRUE to replace an existing file; it is removed first, so the new file never inherits its permissions.

Details

A secret key file is created readable by its owner only (on Windows it takes the folder's permissions). An existing file is never replaced unless `force = TRUE`. R's base64 here is not constant time: read secret key files where timing cannot be observed.

Value

`nqreadkey()`: a list with `alg` (the algorithm name as the library spells it), `kind` ("pub" or "sec") and `key` (a raw vector). `nqwritekey()`: path, invisibly.

See Also

[nqkeygen\(\)](#), [nqsign\(\)](#).

Examples

```
k <- nqkeygen("ML-DSA-44")
f <- tempfile()
nqwritekey(f, k$sk, "ML-DSA-44", "sec")
identical(nqreadkey(f)$key, k$sk)
readLines(f, n = 1)
```

nqrotate

Rotate or revoke a seal key

Description

A key statement (`noqtis-keys-1`, canonical JSON) says that a signing key is replaced by a new one from a given time (`nqrotate()`, signed by both keys) or revoked from then (`nqrevoke()`, signed by the key). [nqcheck\(\)](#) follows statements from a trusted key with its `keys` argument. Both algorithms must be post-quantum: Ed25519 alone is refused.

Usage

```
nqrotate(sk, pk, alg, nsk, npk, nalg, at = Sys.time(), reason = NULL)
```

```
nqrevoke(sk, pk, alg, at = Sys.time(), reason = NULL)
```

Arguments

sk, pk	The key pair the statement is about.
alg	Its signature algorithm.
nsk, npk	The new key pair.
nalg	The new key pair's algorithm.
at	When the statement takes effect: a POSIXct time or seconds since 1970 (UTC); default now.
reason	NULL, or up to 255 bytes of text kept in the statement.

Value

The statement, a raw vector (write it with `writeBin()`).

Examples

```
old <- nqkeygen("ML-DSA-44")
new <- nqkeygen("ML-DSA-65")
s <- nqrotate(old$sk, old$pk, "ML-DSA-44", new$sk, new$pk, "ML-DSA-65",
  reason = "yearly")
cat(rawToChar(s))
```

nqseal

Seal and check a capsule

Description

A capsule is a signed record of the exact files in a folder. `nqseal()` writes `noqtis.json` (the manifest: every regular file with its size, SHA-256 and SHA3-256) and `noqtis.sig` (the signature over the manifest) into the folder. `nqcheck()` checks a folder against its seal and reports every check.

Usage

```
nqseal(
  dir,
  sk,
  pk,
  alg,
  meta = NULL,
  format = if (is.null(inputs)) 1L else 2L,
```

```

    inputs = NULL
)

nqcheck(
    dir,
    pk = NULL,
    anchors = NULL,
    only = NULL,
    lineage = NULL,
    keys = NULL,
    proof = NULL
)

```

Arguments

dir	Path of the folder.
sk	Secret key, a raw vector (as from nqkeygen()).
pk	Public key, a raw vector; for nqseal() it must pair with sk and is written into the seal; for nqcheck() , NULL accepts the key in the seal (the key row then says so), otherwise the seal must be made with this key.
alg	Signature algorithm name, one of the "sig" rows of nqalgs() .
meta	NULL, or one string of JSON text (an object) stored in the manifest under "meta".
format	1 (the default) or 2. Format 2 adds a Merkle tree over the files to the manifest and signs its head, so that nqprove() can show some files without the others.
inputs	NULL, or the folders of the capsules this one was made from (format 2, the default with it). Each is recorded by its id (nqcapid()) under its name: <code>names(inputs)</code> where given, else the folder's last component. Check the inputs first.
anchors	NULL, or a list of up to 32 raw vectors, DER encoded certificates of trusted timestamp authorities; the seal must then carry an RFC 3161 timestamp token that verifies under one of them.
only	NULL, or the paths (as the manifest lists them, / between folders) of the files to check; the others are not read, and <code>partial</code> rows say so.
lineage	NULL, or capsule folders among which the inputs of a format 2 capsule are found by id and checked, with no key pinned, to 64 levels; one lineage row per input.
keys	NULL, or key statements from nqrotate() or nqrevoke() (a list of raw vectors, or a character vector of files), at most 32; they need pk. The seal may then be by a key pk was rotated to, and must carry a verified timestamp from before its key was rotated away or revoked.
proof	NULL, or a proof from nqprove() (a raw vector or the path of a file), checked instead of the folder: its seal, its signature over the manifest's head and each proven entry. <code>dir</code> may then be NULL; given, the proven files in it are checked too. <code>only</code> , <code>lineage</code> and <code>keys</code> do not apply.

Details

The creation time in the manifest is the SOURCE_DATE_EPOCH environment variable when set, else the clock. Sealing refuses a folder that holds a symbolic link or a special file. Capsules are the same files the noqtis command-line tool writes and checks.

The library makes no network requests: an RFC 3161 timestamp is added to a seal by other tools, and nqcheck() verifies it when anchors is given.

Value

nqseal(): the paths of the two files written, invisibly. nqcheck(): a data frame with one row per check and the columns check, ok (logical) and detail; its attribute "ok" is TRUE only when every check passed.

Examples

```
d <- file.path(tempdir(), "capsule")
dir.create(d)
writeLines("hello", file.path(d, "a.txt"))
k <- nqkeygen("ML-DSA-44")
nqseal(d, k$sk, k$pk, "ML-DSA-44", meta = '{"project":"demo"}')
r <- nqcheck(d, k$pk)
r
attr(r, "ok")
writeLines("hello", file.path(d, "a.txt"))
attr(nqcheck(d, k$pk), "ok")
attr(nqcheck(d, k$pk, only = "b.txt"), "ok")
unlink(d, recursive = TRUE)
```

nqsign

Sign and verify

Description

nqsign() signs a message with ML-DSA (FIPS 204), SLH-DSA (FIPS 205), Ed25519 (RFC 8032) or a hybrid ML-DSA+Ed25519, whose signature verifies only when both parts do; nqverify() checks such a signature.

Usage

```
nqsign(sk, msg, alg, ctx = NULL, det = FALSE, ph = NULL, rnd = NULL)
```

```
nqverify(pk, msg, sig, alg, ctx = NULL, ph = NULL)
```

Arguments

sk	Secret key, a raw vector (as from nqkeygen()).
msg	The message: a raw vector, or one string, which is signed as its UTF-8 bytes.
alg	Signature algorithm name, one of the "sig" rows of nqalgs() .
ctx	NULL or a context string (raw vector or string, at most 255 bytes), bound into the signature; verification needs the same one.
det	FALSE (the default) for hedged signing with fresh randomness; TRUE for the deterministic variant (all-zero per-signature randomness).
ph	NULL for the pure algorithm, or a hash name (see nqhash()) for HashML-DSA / HashSLH-DSA, which hash msg with it first.
rnd	NULL, or the standard's per-signature randomness as a raw vector (32 bytes for ML-DSA and the hybrids, n bytes for SLH-DSA, none for Ed25519), for test vectors; it overrides det.
pk	Public key, a raw vector.
sig	Signature, a raw vector.

Value

nqsign(): the signature, a raw vector. nqverify(): TRUE when the signature verifies, FALSE otherwise (including a signature, key or context of the wrong length or encoding).

Examples

```
k <- nqkeygen("ML-DSA-44")
s <- nqsign(k$sk, "hello", "ML-DSA-44")
nqverify(k$pk, "hello", s, "ML-DSA-44")
nqverify(k$pk, "hello", s, "ML-DSA-44")
```

nqver

Version of the bundled noqtis library

Description

Version of the bundled noqtis library

Usage

```
nqver()
```

Value

A string, the version of the C library compiled into this package.

Examples

```
nqver()
```

Index

noqtis-package, [9](#)
nqalgs, [2](#)
nqalgs(), [4](#), [5](#), [12](#), [14](#)
nqcapid, [3](#)
nqcapid(), [12](#)
nqcheck (nqseal), [11](#)
nqcheck(), [3](#), [10](#)
nqdecap (nqencap), [5](#)
nqdsse, [4](#)
nqdsseverify (nqdsse), [4](#)
nqencap, [5](#)
nqencap(), [9](#)
nqfiles (nqcapid), [3](#)
nqhash, [6](#)
nqhash(), [14](#)
nqhmac (nqhash), [6](#)
nqidcard, [7](#)
nqidcheck (nqidcard), [7](#)
nqkeygen, [8](#)
nqkeygen(), [5](#), [10](#), [12](#), [14](#)
nqprove (nqcapid), [3](#)
nqprove(), [12](#)
nqrand, [9](#)
nqreadkey, [9](#)
nqrevoke (nqrotate), [10](#)
nqrevoke(), [12](#)
nqrotate, [10](#)
nqrotate(), [12](#)
nqseal, [11](#)
nqseal(), [3](#)
nqsign, [13](#)
nqsign(), [9](#), [10](#)
nqver, [14](#)
nqverify (nqsign), [13](#)
nqwritekey (nqreadkey), [9](#)

set.seed(), [9](#)

writeBin(), [8](#), [11](#)