Package ‘PKI’

July 2, 2014

Version 0.1-1

Title Public Key Infrastructure for R based on the X.509 standard

Author Simon Urbanek <Simon.Urbanek@r-project.org>

Maintainer Simon Urbanek <Simon.Urbanek@r-project.org>

Depends R (>= 2.9.0), base64enc

Enhances gmp

Description This package provides PKI functions such as verifyig certificates, RSA encription and signing which can be used to build PKI infrastructure and perform cryptographic tasks.

License GPL-2 | GPL-3

URL http://www.rforge.net/PKI

SystemRequirements OpenSSL library

NeedsCompilation yes

Repository CRAN

Date/Publication 2013-02-20 07:45:20

R topics documented:

ASN1 *Functions for handling ASN.1 format (typically DER)*

Description

ASN1.decode decodes ASN.1 binary format into raw format chunks tagged with class types.

ASN1.encode converts structured objects into ASN.1 binary format.

ASN1.item creates an item - basic obejct in structures taht can be encoded using ASN1.encode.

ASN1.type extracts the class type from an ASN.1 item

Usage

```
ASN1.decode(what)
ASN1.encode(what)
ASN1.item(what, type)
ASN1.type(what)
```

Arguments

what	object to decode/encode/query
type	class type of the item (integer value)

Details

This is a suite of low-level tools to deal with ASN.1 (Abstract Syntax Notation One) binary formats DER, BER and CER. The tools were written specifically to handle the various DER-encoded key structures so it provides only a subset of the ASN.1 specification. They are used internally by the PKI poackage.

ASN1.decode decodes the binary representation (as raw vector) into individual items. Sequences are convered into lists, all other objects are retained in their binary form and tagged with the integer class type - which can be obtained using ASN1.type function.

ASN1.encode expects item (or a list of items) either created using ASN1.decode or ASN1.item and cinverts them into DER binary format.

The result of ASN1.encode(ASN1.decode(x)) will be x if x was in DER format.

Value

ASN1.decode returns either one item or a list.

ASN1.encode returns a raw vector in DER format.

ASN1.type returns an integer class type

ASN1.item returns an ASN.1 item object

Note

ASN1.encode uses a fixed buffer for encoding which currently limits the total size of the resulting structre to 1MB.

Only definite length forms are supported. The validity of individual items is not checked.

Author(s)

Simon Urbanek

Examples

```
# generate a small key
key <- PKI.genRSAkey(bits = 512L)

# extract private and public parts in DER format
prv <- PKI.save.key(key, format="DER")
pub <- PKI.save.key(key, private=FALSE, format="DER")

# parse the public key
x <- ASN1.decode(pub)
x
# the second element is the actual key
# as a bit string that's itself in DER
# two integers - modulus and exponent
# Note that this is in fact the pure PKCS#1 key format
ASN1.decode(x[[2]])

# encoding it back should yield the same representation since it is DER
stopifnot(identical(ASN1.encode(x), as.raw(pub)))
```

BIGNUMint *Functions for BIGNUM representation of arbitrarily precise integers*

Description

as.BIGNUMint encodes integer in BIGNUM format as raw vector as used by ASN.1 format.

Usage

```
as.BIGNUMint(what, scalar = TRUE)
```

Arguments

what	representation of an integer or a vector thereof. Currently supported formats include "bigz" objects from the "gmp" package, integers and reals.
scalar	if TRUE then the input is expected to be scalar and only the first element will be used (zero-length vectors raise an error). Otherwise the result will be a list of all converted elements.

Details

The BIGNUM representation as used in ASN.1 is a big-endian encoding of variable length stored in a raw vector. Negative numbers are stored in two-complement's encoding, but are curretnly unsupported by `as.BIGNUMint`.

Value

Raw vector in BIGNUM integer representation.

Note

Unless the input is of class `"bigz"` then 32-bit platforms only support integers up to 32-bit, 64-bit platforms up to 53-bit (when real vectors are used).

Author(s)

Simon Urbanek

Examples

```
as.BIGNUMint(65537)
```

`PKI.crypt` *PKI encryption/decryption functions*

Description

`PKI.encrypt` encrypts a raw vector

`PKI.decrypt` decrypts a raw vector

Usage

```
PKI.encrypt(what, key)
PKI.decrypt(what, key)
```

Arguments

`what`	raw vector to encrypt/decrypt. It must not exceed the key size minus padding
`key`	key to use for encryption/decryption

Value

Raw vector (encrypted/decrypted)

Note

Currently only RSA encryption is supported. Note that the payload should be very small since it must fit into the key size including padding. For example, 1024-bit key can only encrypt 87 bytes, while 2048-bit key can encrypt 215 bytes.

Author(s)

Simon Urbanek

See Also

`PKI.genRSAkey`, `PKI.pubkey`

Examples

```
key <- PKI.genRSAkey(2048)
x <- charToRaw("Hello, world!")
e <- PKI.encrypt(x, key)
y <- PKI.decrypt(e, key)
stopifnot(identical(x, y))
print(rawToChar(y))
```

`PKI.digest` *Compute digest sum based on SHA1 or MD5 hash functions*

Description

`PKI.digest` computes digsest sum based on the hash function specified

Usage

```
PKI.digest(what, hash = c("SHA1", "MD5"))
```

Arguments

`what`	raw vector of bytes to digest
`hash`	type of the hash function. Note that "MD5" should *not* be used for cryptographic purposes as it is not secure

Value

Raw vector containg the hash

Author(s)

Simon Urbanek

See Also

PKI.sign

Examples

```
PKI.digest(as.raw(1:10))
```

PKI.sign *PKI: sign content or verify a signature*

Description

PKI.sign signs content using RSA with the specified hash function

PKI.verify verifies a signature of RSA-signed content

Usage

```
PKI.sign(what, key, hash = c("SHA1", "MD5"), digest)
PKI.verify(what, signature, key, hash = c("SHA1", "MD5"), digest)
```

Arguments

what raw vector: content to sign

key RSA private key to use for signing or RSA public key to use for verification. Use PKI.pubkey to obtain a key to verify from a certificate.

hash hash function to use. "MD5" should not be used unless absolutely needed for compatibility as it is less secure.

digest raw vector: it is possible to supply the digest of the content directly instead of specifying what.

signature raw vector: signature

Details

Objects are signed by computing a hash function digest (typically using SHA1 hash function) and then signing the digest with a RSA key. Verification is done by computing the digest and then comparing the signature to the digest. Private key is needed for signing whereas public key is needed for verification.

Both functions call PKI.digest on what if digest is not specified.

Value

PKI.sign signature (raw vector)

PKI.verify logical: TRUE if the digest and signature match, FALSE otherwise

Author(s)

Simon Urbanek

See Also

PKI.pubkey, PKI.genRSAkey, PKI.digest

Examples

```
key <- PKI.genRSAkey(2048)
x <- charToRaw("My message to sign")
sig <- PKI.sign(x, key)
stopifnot(PKI.verify(x, sig, key))
```

raw2hex *Convert raw vector to string hex representation*

Description

raw2hex convers a raw vector into hexadecimal representation

Usage

```
raw2hex(what, sep, upper = FALSE)
```

Arguments

what	raw vector
sep	optional separator string
upper	logical, if TRUE then upper case letters are used, otherwise any letters will be lower case.

Details

If sep is omitted or NULL then the resulting character vector will have as many elements as the raw vector. Otherwise the elements are concatenated using the specified separator into one character string. This is much more efficient than using paste(raw2hex(x), collapse=sep), but has the same effect.

Value

Character vector with the hexadecimal representation of the raw vector.

Author(s)

Simon Urbanek

Examples

```
raw2hex(PKI.digest(raw(), "SHA1"), "")
raw2hex(PKI.digest(raw(), "MD5"), ":")

## this is jsut a performance comparison and a test that
## raw2hex can handle long strings
x <- as.raw(runif(1e5) * 255.9)
system.time(h1 <- raw2hex(x, " "))
system.time(h2 <- paste(raw2hex(x), collapse=" "))
stopifnot(identical(h1, h2))
```

`RSA` *PKI functions handling RSA keys*

Description

`PKI.load.key` loads an RSA key in PKCS#1 PEM or DER format.

`PKI.save.key` creates a PEM or DER representation of a RSA key.

`PKI.genRSAkey` generates RSA public/private key pair.

`PKI.mkRSApubkey` creates a RSA public key with the supplied modulus and exponent.

`PKI.load.OpenSSH.pubkey` loads public key in OpenSSH format (as used in `.ssh/authorized_keys` file)

Usage

```
PKI.load.key(what, format = c("PEM", "DER"), private, file)
PKI.save.key(key, format = c("PEM", "DER"), private, target)
PKI.genRSAkey(bits = 2048L)
PKI.mkRSApubkey(modulus, exponent=65537L, format = c("DER", "PEM", "key"))
PKI.load.OpenSSH.pubkey(what, first=TRUE, format = c("DER", "PEM", "key"))
```

Arguments

`what` string, raw vector or connection to load the key from

`key` RSA key object

`format` format - PEM is ASCII (essentially base64-encoded DER with header/footer), DER is binary and key means an acutal key object

`private` logical, whether to use the private key (`TRUE`), public key (`FALSE`) or whichever is available (`NA` or missing).

`file` filename to load the key from - `what` and `file` are mutually exclusive

`target` optional connection or a file name to store the result in. If missing, the result is just returned form teh function as either a character vector (PEM) or a raw vector (DER).

`bits` size of the generated key in bits. Must be `2 ^ n` with integer `n > 8`.

`modulus` modulus either as a raw vector (see `as.BIGNUMint`) or `bigz` object (from gmp package) or an integer.

`exponent` exponent either as a raw vector (see `as.BIGNUMint`) or `bigz` object (from gmp package) or an integer.

`first` logical, if `TRUE` only the first key will be used, otherwise the result is a list of keys.

Value

`PKI.load.key`: private or public key object

`PKI.save.key`: raw vector (DER format) or character vector (PEM format).

`PKI.genRSAkey`: private + public key object

`PKI.mkRSApubkey`, `PKI.load.OpenSSH.pubkey`: raw vector (DER format) or character vector (PEM format) or a `"public.key"` object.

Note

The format for private keys in PEM is PKCS#1, but for public keys it is X.509 SubjectPublicKeyInfo (certificate public key). This is consistent with OpenSSL RSA command line tool which uses the same convention.

The OpenSSH format is one line beginning with `"ssh-rsa "`. SSH2 PEM public keys (rfc4716) are supported in `PKI.load.key` and the binary payload is the same as the OpenSSH, only with different wrapping.

Author(s)

Simon Urbanek

See Also

`PKI.encrypt`, `PKI.decrypt`, `PKI.pubkey`

Examples

```
# generate 2048-bit RSA key
key <- PKI.genRSAkey(bits = 2048L)

# extract private and public parts as PEM
priv.pem <- PKI.save.key(key)
pub.pem <- PKI.save.key(key, private=FALSE)
# load back the public key separately
pub.k <- PKI.load.key(pub.pem)

# encrypt with the public key
x <- PKI.encrypt(charToRaw("Hello, world!"), pub.k)
# decrypt with private key
rawToChar(PKI.decrypt(x, key))

# compute SHA1 hash (fingerprint) of the public key
```

```
PKI.digest(PKI.save.key(key, "DER", private=FALSE))

# convert OpenSSH public key to PEM format
PKI.load.OpenSSH.pubkey("ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAuvOXqfZ3pJeWeqyQOIXZwmgM1RBqPUmVx3XgntpA+YtOZjKfu
```

X509 *Public Key Instraftructure (X509) functions*

Description

`PKI.load.cert` creates a certificate obejct from a string, connection or file.

`PKI.verifyCA` verifies a certificate against a given chain of trust.

`PKI.pubkey` extracts public key from a certificate.

Usage

```
PKI.load.cert(what, format = "PEM", file)
PKI.verifyCA(certificate, ca)
PKI.pubkey(certificate)
```

Arguments

`what`	string, raw vector or connection to load teh certificate from
`format`	format used to encode the certificate
`file`	filename to load the certificate from - `what` and `file` are mutually exclusive
`certificate`	a certificate object (as returned by `PKI.load.cert`)
`ca`	a certificate object of the Certificate Authority (CA) or a list of such objects if multiple CAs are involved

Value

`PKI.load.code`: a certificate object

`PKI.verifyCA`: `TRUE` is the certificate can be trusted, `FALSE` otherwise

`PKI.pubkey`: public key object

Author(s)

Simon Urbanek

Examples

```
ca <- PKI.load.cert(file=system.file("certs", "RForge-ca.crt", package="PKI"))
my.cert <- PKI.load.cert(readLines(system.file("certs", "demo.crt", package="PKI")))
PKI.verifyCA(my.cert, ca)
PKI.pubkey(my.cert)
```

Index