# Author: Trevor Perrin

# See the LICENSE file for legal information regarding use of this file.

"""OpenSSL/M2Crypto RSA implementation."""

from .cryptomath import \*

from .rsakey import \*

from .python\_rsakey import Python\_RSAKey

#copied from M2Crypto.util.py, so when we load the local copy of m2

#we can still use it

def password\_callback(v, prompt1='Enter private key passphrase:',

prompt2='Verify passphrase:'):

from getpass import getpass

while 1:

try:

p1=getpass(prompt1)

if v:

p2=getpass(prompt2)

if p1==p2:

break

else:

break

except KeyboardInterrupt:

return None

return p1

if m2cryptoLoaded:

class OpenSSL\_RSAKey(RSAKey):

def \_\_init\_\_(self, n=0, e=0):

self.rsa = None

self.\_hasPrivateKey = False

if (n and not e) or (e and not n):

raise AssertionError()

if n and e:

self.rsa = m2.rsa\_new()

m2.rsa\_set\_n(self.rsa, numberToMPI(n))

m2.rsa\_set\_e(self.rsa, numberToMPI(e))

def \_\_del\_\_(self):

if self.rsa:

m2.rsa\_free(self.rsa)

def \_\_getattr\_\_(self, name):

if name == 'e':

if not self.rsa:

return 0

return mpiToNumber(m2.rsa\_get\_e(self.rsa))

elif name == 'n':

if not self.rsa:

return 0

return mpiToNumber(m2.rsa\_get\_n(self.rsa))

else:

raise AttributeError

def hasPrivateKey(self):

return self.\_hasPrivateKey

def \_rawPrivateKeyOp(self, m):

b = numberToByteArray(m, numBytes(self.n))

s = m2.rsa\_private\_encrypt(self.rsa, bytes(b), m2.no\_padding)

c = bytesToNumber(bytearray(s))

return c

def \_rawPublicKeyOp(self, c):

b = numberToByteArray(c, numBytes(self.n))

s = m2.rsa\_public\_decrypt(self.rsa, bytes(b), m2.no\_padding)

m = bytesToNumber(bytearray(s))

return m

def acceptsPassword(self): return True

def write(self, password=None):

bio = m2.bio\_new(m2.bio\_s\_mem())

if self.\_hasPrivateKey:

if password:

def f(v): return password

m2.rsa\_write\_key(self.rsa, bio, m2.des\_ede\_cbc(), f)

else:

def f(): pass

m2.rsa\_write\_key\_no\_cipher(self.rsa, bio, f)

else:

if password:

raise AssertionError()

m2.rsa\_write\_pub\_key(self.rsa, bio)

s = m2.bio\_read(bio, m2.bio\_ctrl\_pending(bio))

m2.bio\_free(bio)

return s

def generate(bits):

key = OpenSSL\_RSAKey()

def f():pass

key.rsa = m2.rsa\_generate\_key(bits, 3, f)

key.\_hasPrivateKey = True

return key

generate = staticmethod(generate)

def parse(s, passwordCallback=None):

# Skip forward to the first PEM header

start = s.find("-----BEGIN ")

if start == -1:

raise SyntaxError()

s = s[start:]

if s.startswith("-----BEGIN "):

if passwordCallback==None:

callback = password\_callback

else:

def f(v, prompt1=None, prompt2=None):

return passwordCallback()

callback = f

bio = m2.bio\_new(m2.bio\_s\_mem())

try:

m2.bio\_write(bio, s)

key = OpenSSL\_RSAKey()

if s.startswith("-----BEGIN RSA PRIVATE KEY-----"):

def f():pass

key.rsa = m2.rsa\_read\_key(bio, callback)

if key.rsa == None:

raise SyntaxError()

key.\_hasPrivateKey = True

elif s.startswith("-----BEGIN PUBLIC KEY-----"):

key.rsa = m2.rsa\_read\_pub\_key(bio)

if key.rsa == None:

raise SyntaxError()

key.\_hasPrivateKey = False

else:

raise SyntaxError()

return key

finally:

m2.bio\_free(bio)

else:

raise SyntaxError()

parse = staticmethod(parse)