The POW Package

Peter Shannon

The POW Package
by Peter Shannon

Table of Contents

List of Examples

Chapter 1. The POW Module

Introduction

This third major release of POW addresses the most critical missing parts of functionality, X509v3 support. Initially I thought adding support via the OpenSSL code would be the easiest option but this proved to be incorrect mainly due to the way I have chosen to handle the complex data such as `directoryNames` and `generalNames`. It is easier in python to construct complex sets of data using lists and dictionaries than coordinate large numbers of objects and method calls. This is no criticism, it is just extremely easy. Coding complex data such as the `certificatePolicies` coding coding routines in C to handle the data proved laborous and ultimately error prone.

PKIX structures are supported by a few operations on the relevant POW objects and through a Python library which is modelled on the DER encoding rules. Modeling DER does expose some of the complexities of the ASN1 specifications but avoids coding many assumptions into the data structures and the interface for the objects. For an example of overly complex definitions take a look at the `Name` object in RFC3280. It is equally important that modeling DER in the way leads to a library which is trivial to extend to support new objects - simple objects are one liners and complex objects only require the definition of a new constructor.

functionality have been plugged. The `Ssl` class has received several new features relating to security. Other areas have been improved: PRNG support, certificate and CRL signing, certificate chain and client verification. Many bugs have been fixed, and certain parts of code re-written where necessary. I hope you enjoy using POW and please feel free to send me feature requests and bug reports.

Module Functions

Function Prototypes

```
def pemRead(type, string, pass=None):
def derRead(type, string):
def seed(data):
def add(data, entropy):
def readRandomFile(filename):
def writeRandomFile(filename):
def getError():
def clearError():
def addObject(oid, shortName, longName):
```

Function Descriptions

The `pemRead` Function

```
def pemRead(type, string, pass=None):
```

This function attempts to parse the `string` according to the PEM type passed. `type` should be one of the following:

```
RSA_PUBLIC_KEY
RSA_PRIVATE_KEY
X509_CERTIFICATE
X509_CRL
```

`pass` should only be provided if an encrypted `Asymmetric` is being loaded. If the password is incorrect an exception will be raised, if no password is provided and the PEM file is encrypted the user will be prompted. If this is not desirable, always supply a password. The object returned will be and instance of `Asymmetric`, `X509` or `X509Crl`.

The `derRead` Function

```
def derRead(type, string):
```

This function attempts to parse the `string` according to the PEM type passed. `type` should be one of the following:

```
RSA_PUBLIC_KEY
RSA_PRIVATE_KEY
X509_CERTIFICATE
X509_CRL
```

As with the PEM operations, the object returned will be and instance of `Asymmetric`, `X509` or `X509Crl`.

The `seed` Function

```
def seed(data):
```

The `seed` function adds data to OpenSSLs PRNG state. It is often said the hardest part of cryptography is getting good random data, after all if you don't have good random data, a 1024 bit key is no better than a 512 bit key and neither would provide protection from a targeted brute force attack. The `seed` and `add` are very similar, except the entropy of the data is assumed to be equal to the length for `seed`. I final point to be aware of, only systems which support /dev/urandom are automatically seeded. If your system does not support /dev/urandom it is your responsibility to seed OpenSSL's PRNG.

The `add` Function

```
def add(data, entropy):
```

The `add` function adds data to OpenSSLs PRNG state. `data` should be data obtained from a random source and `entropy` is an estimation of the number of random bytes in `data`.

The `readRandomFile` Function

```
def readRandomFile(filename):
```

This function reads a previously saved random state. It can be very useful to improve the quality of random data used by an application. The random data should be added to, using the `add` function, with data from other suitable random sources.

The `writeRandomFile` Function

```
def writeRandomFile(filename):
```

This function writes the current random state to a file. Clearly this function should be used in conjunction with `readRandomFile`.

The `getError` Function

```
def getError():
```

Pops an error off the global error stack and returns it as a string.

The `clearError` Function

```
def clearError():
```

Removes all errors from the global error stack.

The `addObject` Function

```
def addObject(oid, shortName, longName):
```

This function can be used to dynamically add new objects to OpenSSL. The `oid` should be a string of space separated numbers and `shortName` and `longName` are the names of the object, ie 'cn' and 'commonName'.

Module Classes

The `Ssl` Class

This class provides access to the Secure Socket Layer functionality of OpenSSL. It is designed to be a simple as possible to use and is not designed for high performance applications which handle many simultaneous connections. The original motivation for writing this library was to provide a security layer for network agents written in Python, for this application, good performance with multiple concurrent connections is not an issue.

Class Prototypes

```
class Ssl :
    def __init__(protocol=SSLV23METHOD):
    def setFd(descriptor):
```

```
def accept():
def connect():
def write(string):
def read(amount=1024):
def peerCertificate():
def useCertificate(cert):
def useKey(key):
def checkKey():
def clear():
def shutdown():
def getShutdown():
def getCiphers():
def setCiphers(ciphers):
def getCipher():
def setVerifyMode(mode):
```

The `__init__` Method

This constructor creates a new `Ssl` object which will behave as a client or server, depending on the `protocol` value passed. The `protocol` also determines the protocol type and version and should be one of the following:

```
SSLV2_SERVER_METHOD
SSLV2_CLIENT_METHOD
SSLV2_METHOD
SSLV3_SERVER_METHOD
SSLV3_CLIENT_METHOD
SSLV3_METHOD
TLSV1_SERVER_METHOD
TLSV1_CLIENT_METHOD
TLSV1_METHOD
SSLV23_SERVER_METHOD
SSLV23_CLIENT_METHOD
SSLV23_METHOD
```

The `setFd` Method

This function is used to associate a file descriptor with a `Ssl` object. The file descriptor should belong to an open TCP connection. Once this function has been called, calling `useKey` or `useCertificate` will, fail rasing exceptions.

The `accept` Method

This function will attempt the SSL level accept with a client. The `Ssl` object must have been created using a `XXXXX_SERVER_METHOD` or a `XXXXX_METHOD` and this function should only be called after `useKey`, `useCertificate` and `setFd` functions have been called.

Example 1-1. `accept` function usage

```
keyFile = open( 'test/private.key', 'r' )
certFile = open( 'test/cacert.pem', 'r' )

rsa = POW.pemRead( POW.RSA_PRIVATE_KEY, keyFile.read(), 'pass' )
x509 = POW.pemRead( POW.X509_CERTIFICATE, certFile.read() )

keyFile.close()
certFile.close()
```

```
sl = POW.Ssl( POW.SSLV23_SERVER_METHOD )
sl.useCertificate( x509 )
sl.useKey( rsa )

s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
s.bind( ('localhost', 1111) )
s.listen(5)
s2, addr = s.accept()

s.close()

sl.setFd( s2.fileno() )
sl.accept()
print sl.read(1024)
sl.write('Message from server to client...')

s2.close()
```

The `connect` Method

This function will attempt the SSL level connection with a server. The `Ssl` object must have been created using a `XXXXX_CLIENT_METHOD` or a `XXXXX_METHOD` and this function should only be called after `setFd` has already been called.

Example 1-2. `connect` function usage

```
s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
s.connect(('localhost', 1111))

sl = POW.Ssl( POW.SSLV23_CLIENT_METHOD )
sl.setFd( s.fileno() )
sl.connect()
sl.write('Message from client to server...')
print sl.read(1024)
```

The `write` Method

This method writes the `string` to the `Ssl` object, to be read by it's peer. This function is analogous to the `socket` classes `write` function.

The `read` Method

This method reads up to `amount` characters from the `Ssl` object. This function is analogous to the `socket` classes `read` function.

The `peerCertificate` Method

This method returns any peer certificate presented in the initial SSL negotiation or `None`. If a certificate is returned, it will be an instance of `X509`.

The `useCertificate` Method

The parameter `cert` must be an instance of the `X590` class and must be called before `setFd`.

The `useKey` Method

The parameter `key` must be an instance of the `Asymmetric` class and must contain the private key. This function cannot be called after `useKey`.

The `checkKey` Method

This simple method will return 1 if the public key, contained in the X509 certificate this `Ssl` instance is using, matches the private key this `Ssl` instance is using. Otherwise it will return 0.

The `clear` Method

This method will clear the SSL session ready for a new SSL connection. It will not effect the underlying socket.

The `shutdown` Method

This method will issue a `shutdown` signal to it's peer. If this connection's peer has already initiated a shutdown this call will succeed, otherwise it will raise and exception. In order to check the shutdown handshake was successful, `shutdown` must be called again. If no exception is raised, the handshake is complete.

The odd implementation of this function reflects the underlying OpenSSL function, which reflects the SSL protocol. Although rasing an exception is a bit annoying, the alternative, returning true all false will not tell you why the call failed and the exception will, at least that is the theory. Look up the exact meaning of the exceptions in the OpenSSL man page SSL_get_error.

The `getShutdown` Method

This function returns an integer indicating the state of the SSL connection. `SSL_RECIEVED_SHUTDOWN` will be set the if it's peer sends a `shutdown` signal or the underlying socket receives a close notify . The possible values are:

```
SSL_NO_SHUTDOWN
SSL_SENT_SHUTDOWN
SSL_RECIEVED_SHUTDOWN
SSL_SENT_SHUTDOWN | SSL_RECIEVED_SHUTDOWN
```

The `getCiphers` Method

This function returns a list of available ciphers ordered from most favored to least. This function must be called after `setFd`.

The `setCiphers` Method

`setCiphers` can help protect against certain types of attacks which try to coerce the server, client or both to negotiate a weak cipher. `ciphers` should be a list of strings, as produced by `getCiphers` and described in the OpenSSL man page ciphers. `setCiphers` should only be called after `setFd`.

The `getCipher` Method

This function returns the current cipher in use.

The `setVerifyMode` Method

This function sets the behavior of the SSL handshake. The parameter `mode` should be one of the following:

```
SSL_VERIFY_NONE
SSL_VERIFY_PEER
```

See the OpenSSL man page `SSL_CTX_set_verify` for details. This function must be called after `setfd` has been called.

The `X509` Class

This class provides access to a significant proportion of X509 functionality of OpenSSL.

Example 1-3. `x509` class usage

```
privateFile = open('test/private.key', 'r')
publicFile = open('test/public.key', 'r')
certFile = open('test/cacert.pem', 'w')

publicKey = POW.pemRead(POW.RSA_PUBLIC_KEY, publicFile.read())
privateKey = POW.pemRead(POW.RSA_PRIVATE_KEY, privateFile.read(), 'pass')

c = POW.X509()

name = [  ['C', 'GB'], ['ST', 'Hertfordshire'],
          ['O','The House'], ['CN', 'Peter Shannon'] ]

c.setIssuer( name )
c.setSubject( name )
c.setSerial(0)
t1 = POW.pkix.time2utc( time.time() )
t2 = POW.pkix.time2utc( time.time() + 60*60*24*365)
c.setNotBefore(t1)
c.setNotAfter(t2)
c.setPublicKey(publicKey)
c.sign(privateKey)

certFile.write( c.pemWrite() )

privateFile.close()
publicFile.close()
certFile.close()
```

Class Prototypes

```
class X509 :
    def __init__():
    def pemWrite():
    def derWrite():
    def sign(key, digest=MD5_DIGEST):
    def setPublicKey(key):
    def getVersion():
    def setVersion(version):
    def getSerial():
    def setSerial(serial):
```

```
def getIssuer(format=SHORTNAME_FORMAT):
def setIssuer(name):
def getSubject(format=SHORTNAME_FORMAT):
def setSubject(name):
def getNotBefore():
def setNotBefore(time):
def getNotAfter():
def setNotAfter(time):
def addExtension(extensionName, critical, extensionValue):
def clearExtensions():
def countExtensions():
def getExtension(index):
def pprint():
```

The `__init__` Method

This constructor creates a skeletal X509 certificate object. It won't be any use at all until several structures have been created using it's member functions.

The `pemWrite` Method

This method returns a PEM encoded certificate as a string.

The `derWrite` Method

This method returns a DER encoded certificate as a string.

The `sign` Method

This method signs a certificate with a private key. See the example for the methods which should be invoked before signing a certificate. `key` should be an instance of `Asymmetric` containing a private key. The optional parameter `digest` indicates which digest function should be used to compute the hash to be signed, it should be one of the following:

```
MD2_DIGEST
MD5_DIGEST
SHA_DIGEST
SHA1_DIGEST
RIPEMD160_DIGEST
```

The `setPublicKey` Method

This method sets the public key for this certificate object. The parameter `key` should be an instance of `Asymmetric` containing a public key.

The `getVersion` Method

This method returns the version number from the version field of this certificate.

The `setVersion` Method

This method sets the version number in the version field of this certificate. `version` should be an integer.

The `getSerial` Method

This method get the serial number in the serial field of this certificate.

The `setSerial` Method

This method sets the serial number in the serial field of this certificate. `serial` should ba an integer.

The `getIssuer` Method

This method returns a tuple containing the issuers name. Each element of the tuple is a tuple with 2 elements. The first tuple is an object name and the second is it's value. Both issuer and subject are names distinguished normally composed of a small number of objects:

```
c or countryName
st or stateOrProvinceName
o or organizationName
l or localityName
ou or organizationalUnitName
cn or commonName
```

The data type varies from one object to another, however, all the common objects are strings. It would be possible to specify any kind of object but that would certainly adversely effect portability and is not recommended.

The `setIssuer` Method

This method is used to set the issuers name. `name` can be comprised of lists or tuples in the format described in the `getissuer` method.

The `getSubject` Method

This method returns a tuple containing the subjects name. See `getIssuer` for a description of the returned object's format.

The `setSubject` Method

This method is used to set the subjects name. `name` can be comprised of lists or tuples in the format described in the `getIssuer` method.

The `getNotBefore` Method

In a change from previous releases, for reasons of portability and to avoid hard to fix issues with problems in unreliable time functions, this function returns a UTCTime string. You can use the function `time2utc` to convert to a string if you like and `utc2time` to back.

The `setNotBefore` Method

In a change from previous releases, for reasons of portability and to avoid hard to fix issues with problems in unreliable time functions, this accepts one parameter, a UTCTime string. You can use the function `time2utc` to convert to a string if you like and `utc2time` to back.

The `getNotAfter` Method

In a change from previous releases, for reasons of portability and to avoid hard to fix issues with problems in unreliable time functions, this function returns a UTCTime string. You can use the function `time2utc` to convert to a string if you like and `utc2time` to back.

The `setNotAfter` Method

In a change from previous releases, for reasons of portability and to avoid hard to fix issues with problems in unreliable time functions, this accepts one parameter, a UTCTime string. You can use the function `time2utc` to convert to a string if you like and `utc2time` to back.

The `addExtension` Method

This method adds an extension to this certificate. `extensionName` should be the of the extension. `critical` should an integer, 1 for true and 0 for false. `extensionValue` should be a string, DER encoded value of the extension. The name of the extension must be correct according to OpenSSL and can be checked in the `objects.h` header file, part of the OpenSSL source distribution. In the majority of cases they are the same as those defined in `POW._oids` but if you do encounter problems is may be worth checking.

Example 1-4. `addExtension` method usage

```
basic = POW.pkix.BasicConstraints()
basic.set([1,5])
serverCert.addExtension( 'basicConstraints', 0, basic.toString())
```

The `clearExtensions` Method

This method clears the structure which holds the extension for this certificate.

The `countExtensions` Method

This method returns the size of the structure which holds the extension for this certificate.

The `getExtension` Method

This method returns a tuple equivalent the parameters of `addExtension`.

The `pprint` Method

This method returns a formatted string showing the information held in the certificate.

The `X509Crl` Class

This class provides access to OpenSSL X509 CRL management facilities.

Class Prototypes

```
class X509Crl :
    def pemWrite():
    def derWrite():
    def getVersion():
    def setVersion(version):
```

```
def getIssuer(format=SHORTNAME_FORMAT):
def setIssuer(name):
def getThisUpdate():
def setThisUpdate(time):
def getNextUpdate():
def setNextUpdate(time):
def getRevoked():
def setRevoked(revoked):
def verify(key):
def sign(key, digest=MD5_DIGEST):
def addExtension(extensionName, critical, extensionValue):
def clearExtensions():
def countExtensions():
def getExtension(index):
def pprint():
```

The `pemWrite` Method

This method returns a PEM encoded CRL as a string.

The `derWrite` Method

This method returns a DER encoded CRL as a string.

The `getVersion` Method

This method returns the version number from the version field of this CRL.

The `setVersion` Method

This method sets the version number in the version field of this CRL. `version` should be an integer.

The `getIssuer` Method

This method returns a tuple containing the issuers name. See the `getIssuer` method of `X509` for more details.

The `setIssuer` Method

This method is used to set the issuers name. `name` can be comprised of lists or tuples in the format described in the `getIssuer` method of `X509`.

The `getThisUpdate` Method

In a change from previous releases, for reasons of portability and to avoid hard to fix issues with problems in unreliable time functions, this function returns a UTCTime string. You can use the function `time2utc` to convert to a string if you like and `utc2time` to back.

The `setThisUpdate` Method

In a change from previous releases, for reasons of portability and to avoid hard to fix issues with problems in unreliable time functions, this accepts one parameter, a UTCTime string. You can use the function `time2utc` to convert to a string if you like and `utc2time` to back.

The `getNextUpdate` Method

In a change from previous releases, for reasons of portability and to avoid hard to fix issues with problems in unreliable time functions, this function returns a UTCTime string. You can use the function `time2utc` to convert to a string if you like and `utc2time` to back.

The `setNextUpdate` Method

In a change from previous releases, for reasons of portability and to avoid hard to fix issues with problems in unreliable time functions, this accepts one parameter, a UTCTime string. You can use the function `time2utc` to convert to a string if you like and `utc2time` to back.

The `getRevoked` Method

This method returns a tuple of `X509Revoked` objects described in the CRL.

Example 1-5. `getRevoked` function usage

```
publicFile = open('test/public.key', 'r')
crlFile = open('test/crl.pem', 'r')

publicKey = POW.pemRead(POW.RSA_PUBLIC_KEY, publicFile.read())

crl = POW.pemRead( POW.X509_CRL, crlFile.read() )

print crl.pprint()
if crl.verify( publicKey ):
   print 'signature ok!'
else:
   print 'signature not ok!'

revocations = crl.getRevoked()
for revoked in revocations:
   print 'serial number:', revoked.getSerial()
   print 'date:', time.ctime( revoked.getDate()[0] )

publicFile.close()
crlFile.close()
```

The `setRevoked` Method

This method sets the sequence of revoked certificates in this CRL. `revoked` should be a list or tuple of `X509Revoked`.

Example 1-6. `setRevoked` function usage

```
privateFile = open('test/private.key', 'r')
publicFile = open('test/public.key', 'r')
crlFile = open('test/crl.pem', 'w')

publicKey = POW.pemRead(POW.RSA_PUBLIC_KEY, publicFile.read())
privateKey = POW.pemRead(POW.RSA_PRIVATE_KEY, privateFile.read(), 'pass')

crl = POW.X509Crl()

name = [  ['C', 'GB'], ['ST', 'Hertfordshire'],
```

```
          ['O','The House'], ['CN', 'Peter Shannon'] ]

t1 = POW.pkix.time2utc( time.time() )
t2 = POW.pkix.time2utc( time.time() + 60*60*24*365)
crl.setIssuer( name )
rev = [  POW.X509Revoked(3, t1),
         POW.X509Revoked(4, t1),
         POW.X509Revoked(5, t1)    ]

crl.setRevoked( rev )
crl.setThisUpdate(t1)
crl.setNextUpdate(t2)
crl.sign(privateKey)

crlFile.write( crl.pemWrite() )

privateFile.close()
publicFile.close()
crlFile.close()
```

The `verify` Method

The `X509Crl` method `verify` is based on the `X509_CRL_verify` function. Unlike the `X509` function of the same name, this function simply checks the CRL was signed with the private key which corresponds the parameter `key`. `key` should be an instance of `Asymmetric` and contain a public key.

The `sign` Method

`key` should be an instance of `Asymmetric` and contain a private key. `digest` indicates which digest function should be used to compute the hash to be signed, it should be one of the following:

```
MD2_DIGEST
MD5_DIGEST
SHA_DIGEST
SHA1_DIGEST
RIPEMD160_DIGEST
```

The `addExtension` Method

This method adds an extension to this CRL. `extensionName` should be the of the extension. `critical` should an integer, 1 for true and 0 for clase. `extensionValue` should be a string, DER encoded value of the extension. The name of the extension must be correct according to OpenSSL and can be checkd in the `objects.h` header file, part of the OpenSSL source distrobution. In the majority of cases they are the same as those defined in `POW._oids` but if you do encounter problems is may be worth checking.

Example 1-7. `addExtension` method usage

```
oids = POW.pkix.OidData()
o2i = oids.obj2oid

n1 = ('directoryName',  (  (( o2i('countryName'), ('printableString', 'UK') ),),
                           (( o2i('stateOrProvinceName'), ('printableString', 'Her
                           (( o2i('organizationName'), ('printableString', 'The Ho
                           (( o2i('commonName'), ('printableString', 'Shannon Work
```

```
n2 = ('rfc822Name', 'peter_shannon@yahoo.com')
n3 = ('uri', 'http://www.p-s.org.uk')
n4 = ('iPAddress', (192,168,100,51))

issuer = POW.pkix.IssuerAltName()
issuer.set([n1,n2,n3,n4])
crl.addExtension( 'issuerAltName', 0, issuer.toString() )
```

The `clearExtensions` Method

This method clears the structure which holds the extension for this CRL.

The `countExtensions` Method

This method returns the size of the structure which holds the extension for this CRL.

The `getExtension` Method

This method returns a tuple equivalent the parameters of `addExtension`.

The `pprint` Method

This method returns a formatted string showing the information held in the CRL.

The `X509Revoked` Class

This class provides a container for details of a revoked certificate. It normally would only be used in association with a CRL, its not much use by itself. Indeed the only reason this class exists is because in the future POW is likely to be extended to support extensions for certificates, CRLs and revocations. `X509Revoked` existing as an object in its own right will make adding this support easier, while avoiding backwards compatibility issues.

Class Prototypes

```
class X509Revoked :
    def __init__(serial, date):
    def getDate():
    def setDate(time):
    def getSerial():
    def setSerial(serial):
    def addExtension(extensionName, critical, extensionValue):
    def clearExtensions():
    def countExtensions():
    def getExtension(index):
```

The `__init__` Method

This constructor builds a X509 Revoked structure. `serial` should be an integer and `date` should be and UTCTime string.

The `getDate` Method

In a change from previous releases, for reasons of portability and to avoid hard to fix issues with problems in unreliable time functions, this function returns a UTCTime string. You can use the function `time2utc` to convert to a string if you like and `utc2time` to back.

The `setDate` Method

In a change from previous releases, for reasons of portability and to avoid hard to fix issues with problems in unreliable time functions, this accepts one parameter, a UTCTime string. You can use the function `time2utc` to convert to a string if you like and `utc2time` to back.

The `getSerial` Method

This method gets the serial number in the serial field of this object.

The `setSerial` Method

This method sets the serial number in the serial field of this object. `serial` should be an integer.

The `addExtension` Method

This method adds an extension to this revocation. `extensionName` should be the of the extension. `critical` should an integer, 1 for true and 0 for clase. `extensionValue` should be a string, DER encoded value of the extension. The name of the extension must be correct according to OpenSSL and can be checkd in the `objects.h` header file, part of the OpenSSL source distrobution. In the majority of cases they are the same as those defined in `POW._oids` but if you do encounter problems is may be worth checking.

Example 1-8. `addExtension` method usage

```
reason = POW.pkix.CrlReason()
reason.set(1)
revocation.addExtension( 'CRLReason', 0, reason.toString() )
```

The `clearExtensions` Method

This method clears the structure which holds the extension for this revocation.

The `countExtensions` Method

This method returns the size of the structure which holds the extension for this revocation.

The `getExtension` Method

This method returns a tuple equivalent the parameters of `addExtension`.

The `X509Store` Class

This class provides preliminary access to OpenSSL X509 verification facilities.

Example 1-9. `x509_store` class usage

```
store = POW.X509Store()

caFile = open( 'test/cacert.pem', 'r' )
ca = POW.pemRead( POW.X509_CERTIFICATE, caFile.read() )
caFile.close()

store.addTrust( ca )

certFile = open( 'test/foocom.cert', 'r' )
x509 = POW.pemRead( POW.X509_CERTIFICATE, certFile.read() )
certFile.close()

print x509.pprint()

if store.verify( x509 ):
   print 'Verified certificate!.'
else:
   print 'Failed to verify certificate!.'
```

Class Prototypes

```
class X509Store :
    def __init__():
    def verify(certificate):
    def verifyChain(certificate, chain):
    def addTrust(cert):
    def addCrl(crl):
```

The `__init__` Method

This constructor takes no arguments. The `X509Store` returned cannot be used for verifying certificates until at least one trusted certificate has been added.

The `verify` Method

The `X509Store` method `verify` is based on the `X509_verify_cert`. It handles certain aspects of verification but not others. The certificate will be verified against `notBefore`, `notAfter` and trusted certificates. It crucially will not handle checking the certificate against CRLs. This functionality will probably make it into OpenSSL 0.9.7.

The `verifyChain` Method

The `X509Store` method `verifyChain` is based on the `X509_verify_cert` but is initialised with a `X509` object to verify and list of `X509` objects which form a chain to a trusted certificate. Certain aspects of the verification are handled but not others. The certificates will be verified against `notBefore`, `notAfter` and trusted certificates. It crucially will not handle checking the certificate against CRLs. This functionality will probably make it into OpenSSL 0.9.7.

This may all sound quite straight forward but determining the certificate associated with the signature on another certificate can be very time consuming. The management aspects of certificates are addressed by various V3 extensions which are not currently supported.

The `addTrust` Method

This method adds a new certificate to the store to be used in the verification process. `cert` should be an instance of `X509`. Using trusted certificates to manage verification is relatively primitive, more sophisticated systems can be constructed at an application level by by constructing certificate chains to verify.

The `addCrl` Method

This method adds a CRL to a store to be used for verification. `crl` should be an instance of `X509Crl`. Unfortunately, the current stable release of OpenSSL does not support CRL checking for certificate verification. This functionality will probably make it into OpenSSL 0.9.7, until it does this function is useless and CRL verification must be implemented by the application.

The `Digest` Class

This class provides access to the digest functionality of OpenSSL. It emulates the digest modules in the Python Standard Library but does not currently support the `hexdigest` function.

Example 1-10. `digest` class usage

```
plain_text = 'Hello World!'
sha1 = POW.Digest( POW.SHA1_DIGEST )
sha1.update( plain_text )
print ' Plain text: Hello World! =>', sha1.digest()
```

Class Prototypes

```
class Digest :
    def __init__(type):
    def update(data):
    def copy():
    def digest():
```

The `__init__` Method

This constructor creates a new `Digest` object. The parameter `type` specifies what kind of digest to create and should be one of the following:

```
MD2_DIGEST
MD5_DIGEST
SHA_DIGEST
SHA1_DIGEST
RIPEMD160_DIGEST
```

The `update` Method

This method updates the internal structures of the `Digest` object with `data`. `data` should be a string.

The `copy` Method

This method returns a copy of the `Digest` object.

The `digest` Method

This method returns the digest of all the data which has been processed. This function can be called at any time and will not effect the internal structure of the `digest` object.

The `Hmac` Class

This class provides access to the HMAC functionality of OpenSSL. HMAC's are a variant on digest based MACs, which have the interesting property of a provable level of security. HMAC is discussed further in RFC 2104.

Class Prototypes

```
class Hmac :
    def __init__(type, key):
    def update(data):
    def copy():
    def mac():
```

The `__init__` Method

This constructor creates a new `Hmac` object. The parameter `key` should be a string and `type` should be one of the following:

```
MD2_DIGEST
MD5_DIGEST
SHA_DIGEST
SHA1_DIGEST
RIPEMD160_DIGEST
```

The `update` Method

This method updates the internal structures of the `Hmac` object with `data`. `data` should be a string.

The `copy` Method

This method returns a copy of the `Hmac` object.

The `mac` Method

This method returns the MAC of all the data which has been processed. This function can be called at any time and will not effect the internal structure of the `Hmac` object.

The `Symmetric` Class

This class provides access to all the symmetric ciphers in OpenSSL. Initialisation of the cipher structures is performed late, only when `encryptInit` or `decryptInit` is called, the constructor only records the cipher type. It is possible to reuse the `Symmetric` objects by calling `encryptInit` or `decryptInit` again.

Example 1-11. `Symmetric` class usage

```
passphrase = 'my silly passphrase'
md5 = POW.Digest( POW.MD5_DIGEST )
md5.update( passphrase )
```

```
password = md5.digest()[:8]

plaintext = 'cast test message'
cast = POW.Symmetric( POW.CAST5_CFB )
cast.encryptInit( password )
ciphertext = cast.update(plaintext) + cast.final()
print 'Cipher text:', ciphertext

cast.decryptInit( password )
out = cast.update( ciphertext ) + cast.final()
print 'Deciphered text:', out
```

Class Prototypes

```
class Symmetric :
    def __init__(type):
    def encryptInit(key, initialvalue=''):
    def decryptInit(key, initialvalue=''):
    def update(data):
    def final(size=1024):
```

The `__init__` Method

This constructor creates a new `Symmetric` object. The parameter `type` specifies which kind of cipher to create. `type` should be one of the following:

```
DES_ECB       IDEA_CBC
DES_EDE       RC2_ECB
DES_EDE3      RC2_CBC
DES_CFB       RC2_40_CBC
DES_EDE_CFB   RC2_CFB
DES_EDE3_CFB  RC2_OFB
DES_OFB       BF_ECB
DES_EDE_OFB   BF_CBC
DES_EDE3_OFB  BF_CFB
DES_CBC       BF_OFB
DES_EDE_CBC   CAST5_ECB
DES_EDE3_CBC  CAST5_CBC
DESX_CBC      CAST5_CFB
RC4           CAST5_OFB
RC4_40        RC5_32_12_16_CBC
IDEA_ECB      RC5_32_12_16_CFB
IDEA_CFB      RC5_32_12_16_ECB
IDEA_OFB      RC5_32_12_16_OFB
```

Please note your version of OpenSSL might not have been compiled with all the ciphers listed above. If that is the case, which is very likely if you are using a stock binary, the unsuported ciphers will not even be in the module namespace.

The `encryptInit` Method

This method sets up the cipher object to start encrypting a stream of data. The first parameter is the key used to encrypt the data. The second, the `initialvalue` serves a similar purpose the the salt supplied to the Unix `crypt` function. The `initialvalue` is normally chosen at random and often transmitted with the encrypted data, its purpose is to prevent two identical plain texts resulting in two identical cipher texts.

The `decryptInit` Method

This method sets up the cipher object to start decrypting a stream of data. The first value must be the key used to encrypt the data. The second parameter is the `initialvalue` used to encrypt the data.

The `update` Method

This method is used to process the bulk of data being encrypted or decrypted by the cipher object. `data` should be a string.

The `final` Method

Most ciphers are block ciphers, that is they encrypt or decrypt a block of data at a time. Often the data being processed will not fill an entire block, this method processes these half-empty blocks. A string is returned of a maximum length `size`.

The `Asymmetric` Class

This class provides access to RSA asymmetric ciphers in OpenSSL. Other ciphers will probably be supported in the future but this is not a priority.

Class Prototypes

```
class Asymmetric :
    def __init__(ciphertype=RSA_CIPHER, keylength=1024):
    def pemWrite(keytype, ciphertype=None, passphrase=None):
    def derWrite(keytype):
    def publicEncrypt(plaintext):
    def publicDecrypt(ciphertext):
    def privateEncrypt(plaintext):
    def privateDecrypt(ciphertext):
    def sign(digesttext, digesttype):
    def verify(signedtext, digesttext, digesttype):
```

The `__init__` Method

This constructor builds a new cipher object. Only RSA ciphers are currently support, so the first argument should always be `RSA_CIPHER`. The second argument, `keylength`, is normally 512, 768, 1024 or 2048. Key lengths as short as 512 bits are generally considered weak, and can be cracked by determined attackers without tremendous expense.

Example 1-12. `asymmetric` class usage

```
privateFile = open('test/private.key', 'w')
publicFile = open('test/public.key', 'w')

passphrase = 'my silly passphrase'
md5 = POW.Digest( POW.MD5_DIGEST )
md5.update( passphrase )
password = md5.digest()

rsa = POW.Asymmetric( POW.RSA_CIPHER, 1024 )
privateFile.write( rsa.pemWrite(
         POW.RSA_PRIVATE_KEY, POW.DES_EDE3_CFB, password ) )
publicFile.write( rsa.pemWrite( POW.RSA_PUBLIC_KEY ) )
```

```
privateFile.close()
publicFile.close()
```

The `pemWrite` Method

This method is used to write `Asymmetric` objects out as strings. The first argument should be either `RSA_PUBLIC_KEY` or `RSA_PRIVATE_KEY`. Private keys are often saved in encrypted files to offer extra security above access control mechanisms. If the `keytype` is `RSA_PRIVATE_KEY` a `ciphertype` and `passphrase` can also be specified. The `ciphertype` should be one of those listed in the `Symmetric` class section.

The `derWrite` Method

This method is used to write `Asymmetric` objects out as strings. The first argument should be either `RSA_PUBLIC_KEY` or `RSA_PRIVATE_KEY`.

The `publicEncrypt` Method

This method is used to encrypt the `plaintext` using a public key. It should be noted; in practice this function would be used almost exclusively to encrypt symmetric cipher keys and not data since asymmetric cipher operations are very slow.

The `publicDecrypt` Method

This method is used to decrypt the `ciphertext` which has been encrypted using the corresponding private key and the `privateEncrypt` function.

The `privateEncrypt` Method

This method is used to encrypt the `plaintext` using a private key. It should be noted; in practice this function would be used almost exclusively to encrypt symmetric cipher keys and not data since asymmetric cipher operations are very slow.

The `privateDecrypt` Method

This method is used to decrypt ciphertext which has been encrypted using the corresponding public key and the `publicEncrypt` function.

The `sign` Method

This method is used to produce a signed digest text. This instance of `Asymmetric` should be a private key used for signing. The parameter `digesttext` should be a digest of the data to protect against alteration and finally `digesttype` should be one of the following:

```
MD2_DIGEST
MD5_DIGEST
SHA_DIGEST
SHA1_DIGEST
RIPEMD160_DIGEST
```

If the procedure was successful, a string containing the signed digest is returned.

The `verify` Method

This method is used to verify a signed digest text.

Example 1-13. `verify` method usage

```
plain_text = 'Hello World!'
print ' Plain text:', plain_text
digest = POW.Digest( POW.RIPEMD160_DIGEST )
digest.update( plain_text )
print ' Digest text:', digest.digest()

privateFile = open('test/private.key', 'r')
privateKey = POW.pemRead( POW.RSA_PRIVATE_KEY, privateFile.read(), 'pass' )
privateFile.close()
signed_text =  privateKey.sign(digest.digest(), POW.RIPEMD160_DIGEST)
print ' Signed text:', signed_text

digest2 = POW.Digest( POW.RIPEMD160_DIGEST )
digest2.update( plain_text )
publicFile = open('test/public.key', 'r')
publicKey = POW.pemRead( POW.RSA_PUBLIC_KEY, publicFile.read() )
publicFile.close()
if publicKey.verify( signed_text, digest2.digest(), POW.RIPEMD160_DIGEST ):
   print 'Signing verified!'
else:
   print 'Signing gone wrong!'
```

The parameter `signedtext` should be a signed digest text. This instance of `Asymmetric` should correspond to the private key used to sign the digest. The parameter `digesttext` should be a digest of the same data used to produce the `signedtext` and finally `digesttype` should be one of the following:

```
MD2_DIGEST
MD5_DIGEST
SHA_DIGEST
SHA1_DIGEST
RIPEMD160_DIGEST
```

If the procedure was successful, 1 is returned, otherwise 0 is returned.

Chapter 2. The POW.pkix Module

Introduction

This module is a solution to reading and writing X509v3 written purely in Python. It does use limited facilities from POW for signing and verifying but these could be replaced easily. It is an abstract module and to use it successfully RFC3280 should be referred to as well as the sourcecode where necessary. The correct use of many extensions often not clear from the definitions alone. Do refer to the RFC for details.

Each constructed objects defined in the RFC is built from primitives defined by the ASN1 recommedations. Not all ASN1 primitive are available but all those required for X509v3 should be. The implementation is more or less complete for DER encoding the only caveat, aside from a few missing objects, is the behaviour of `SET` objects and `SET OF` objects. The order the objects are written in should be determined at runtime by sorting their tags but this library does not do this. For X509 it isn't really necessary since all the `Set` objects are simple and the order they are written in is defined by the object's constructor.

Every documented object in this module supports the functions documented for `_GeneralObject`. In general the function will only be documented in descendant classes if the class changes the behaviour significantly from its ancestor. This would normally be `_GeneralObject` or `Sequence`.

Module Functions

Function Prototypes

```
def utc2time(time):
def time2utc(time):
def gen2time(time):
def time2gen(time):
```

Function Descriptions

The `utc2time` Function

```
def utc2time(time):
```

This is a helper function for turning a UTCTime string into an integer. It isn't built into the encoder since the various functions which are used to manipulate the tm structure are notoriously unreliable.

The `time2utc` Function

```
def time2utc(time):
```

This is a helper function for turning an integer into a UTCTime string. It isn't built into the encoder since the various functions which are used to manipulate the tm structure are notoriously unreliable.

The `gen2time` Function

```
def gen2time(time):
```

This is a helper function for turning a GeneralizedTime string into an integer. It isn't built into the encoder since the various functions which are used to manipulate the tm structure are notoriously unreliable.

The `time2gen` Function

```
def time2gen(time):
```

This is a helper function for turning an integer into a GeneralizedTime string. It isn't built into the encoder since the various functions which are used to manipulate the tm structure are notoriously unreliable.

Module Classes

The `_GeneralObject` Class

`_GeneralObject` is the basis for all DER objects, primitive or constructed. It defines the basic behaviour of an object which is serialised using the tag, length and value approach of DER. It is unlikely you would ever want to instantiate one of these directly but I include a description since many primatives don't override much of `_GeneralObject`'s functions.

Class Prototypes

```
class _GeneralObject :
    def __init__(normclass, normform, normnumber, encRoutine, decRoutine, optional=0, de
    def reset():
    def set(value):
    def get():
    def implied(impclass, impform, impnumber):
    def read(io):
    def write(io):
    def toString():
    def fromString():
```

The `__init__` Method

`normclass` is the class of the object, ei: universal, application, context or private. `normform` is the form of the object, ei primitive or constructed. `normnumber` is the tag number of the object. `encRoutine` is a function which takes a value and encodes it according the appropriate DER rules. `decRoutine` is a function which reads a string value and returns a value which is more useful in Python. `optional` is a boolean indicating if this object is optional. The final parameter, `default` is the base 64 encoded DER value, which should be used as the default in leu of a value to read or incase it is unset.

The `reset` Method

This function re-initialises the object, clearing the value or setting it to any default.

The `set` Method

This dosn't do much except store `value`, presumably prior to writing the object. The correct values to use would be determined by the encoder or decoder this class is instantiated with. Be careful, there is some flexibility in setting objects so you might find that once the object has been written and read back in the value isn't identical. A good example would be anything which contains a sequence(list or tuple), all sequence objects are returned as tuples.

The `get` Method

Gets the value stored presumably after reading the object.

The `implied` Method

This function is used to change how the tag is written or read for a particular object and should be called in the constructor for derived objects. If you have an example of the structure you need to process, Pete Gutmann's excellent dumpasn1 can be invaluable for debugging objects.

The `read` Method

`io` should be a file like object. If the object being read matches the expected class, form and tag the value is read and decoded using `decRoutine`. Else, if it has a default that is read and stored.

The return value of this function does not indicate success but whether this TLV was processed successfully. This bahaviour is vital for processing constructed types since the object may be optional or have a default. Failure to decode would be indicated by an exception.

The `write` Method

If this object has not been set and is not optional and dosn't have a default, a `DerError` exception will be raised

If no value has been set and this object is optional, nothing is written. If this object's value is equal to the default, nothing is written as stipulated by DER. Otherwise the value is encoded and written.

The `toString` Method

Encodes the value in DER and returns it as a string.

The `fromString` Method

Decodes the string and sets the value of this object.

The `Boolean` Class

This object represents the ASN1 BOOLEAN type. It can be set with any object which can be tested for truth.

Class Prototypes

```
class Boolean (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `Integer` Class

This object represents the ASN1 INTEGER type. It should be set with a Python integer.

Class Prototypes

```
class Integer (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `BitString` Class

This object represents the ASN1 BIT STRING type. It should be set with a sequence of integers. A non-zero number will set the bit, zero will leave the bit unset.

Class Prototypes

```
class BitString (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `AltBitString` Class

This object represents the ASN1 BIT STRING type. It differs from the first `BitString` in that it’s coding routines treat values as binary data and do not interpret the data in any way. Some application treat the `BIT STRING` in the same way as `OCTET STRING` type, hence this extra object.

Class Prototypes

```
class AltBitString (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `OctetString` Class

This object represents the ASN1 OCTET STRING type. This object can be set with any binary data.

Class Prototypes

```
class OctetString (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `Null` Class

This object represents the ASN1 NULL type. There is no point in setting this object, the value will always be ignored when it is written out.

Class Prototypes

```
class Null (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `Oid` Class

This object represents the ASN1 OID type. This object should be set with a list or tuple of integers defining an objects oid. Please note that the first three arcs have a restricted set of values, so encoding (5, 3, 7, 1) will produce bad results.

Class Prototypes

```
class Oid (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `Enum` Class

This object represents the ASN1 ENUM type. This should be set using a Python integer, the meaning should be described in the ASN1 document for the object you are encoding.

Class Prototypes

```
class Enum (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `Utf8String` Class

This object represents the ASN1 UTF8String type. This object should be set with a string. It is up to the application to ensure it only contains valid characters for this type.

Class Prototypes

```
class Utf8String (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `NumericString` Class

This object represents the ASN1 NumericString type. This should object should be set with a string. It is up to the application to ensure it only contains valid characters for this type.

Class Prototypes

```
class NumericString (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `PrintableString` Class

This object represents the ASN1 PrintableString type. This should object should be set with a string. It is up to the application to ensure it only contains valid characters for this type.

Class Prototypes

```
class PrintableString (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `T61String` Class

This object represents the ASN1 T61String type. This object should be set with a string. It is up to the application to ensure it only contains valid characters for this type.

Class Prototypes

```
class T61String (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `VideotexString` Class

This object represents the ASN1 VideotexString type. This should object should be set with a string. It is up to the application to ensure it only contains valid characters for this type.

Class Prototypes

```
class VideotexString (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `IA5String` Class

This object represents the ASN1 IA5String type. This object should be set with a string. It is up to the application to ensure it only contains valid characters for this type.

Class Prototypes

```
class IA5String (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `UtcTime` Class

This object represents the ASN1 UTCTime type. This object should be set with a string of the general format YYMMDDhhmmssZ. The helper functions `time2utc` and `utc2time` can be used to handle the conversion from an integer to a string and back.

Class Prototypes

```
class UtcTime (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `GeneralizedTime` Class

This object represents the ASN1 GeneralizedTime type. This object should be set with a string of the general format YYYYMMDDhhmmssZ. The helper functions `time2utc` and `utc2time` can be used to handle the conversion from an integer to a string and back.

Class Prototypes

```
class GeneralizedTime (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `GraphicString` Class

This object represents the ASN1 GraphicString type. This should object should be set with a string. It is up to the application to ensure it only contains valid characters for this type.

Class Prototypes

```
class GraphicString (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `VisibleString` Class

This object represents the ASN1 VisibleString type. This should object should be set with a string. It is up to the application to ensure it only contains valid characters for this type.

Class Prototypes

```
class VisibleString (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `GeneralString` Class

This object represents the ASN1 GeneralString type. This should object should be set with a string. It is up to the application to ensure it only contains valid characters for this type.

Class Prototypes

```
class GeneralString (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `UniversalString` Class

This object represents the ASN1 UniversalString type. This should object should be set with a string. It is up to the application to ensure it only contains valid characters for this type.

Class Prototypes

```
class UniversalString (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `BmpString` Class

This object represents the ASN1 BMPString type. This object should be set with a string. It is up to the application to ensure it only contains valid characters for this type.

Class Prototypes

```
class BmpString (_GeneralObject):
    def __init__(optional=0, default=”):
```

The `Sequence` Class

This object represents the ASN1 SEQUENCE type.

Class Prototypes

```
class Sequence (_GeneralObject):
    def __init__(contents, optional=0, default=”):
    def readContents(io, contents):
    def read(io):
    def write(file):
    def set(values):
    def get():
```

The `__init__` Method

The `contents` should be a list or tuple containing the contents of the sequence. Two important members are initialised this this constructor. First `self.next` this is used to keep track of which TLVs in this sequence has been read succesfully. The second, `self.contents` should be set to the list of objects stored in this sequence. Note that the order they are specified in is the order in which they are written or read.

The `readContents` Method

This function implements basic SEQUENCE like reading behaviour. It will attempt to read each of the objects in `contents` in turn from `io`. It exists as a function, separate from `read` for the benefit of the SEQUENCE OF implementation.

The TLV of this SEQUENCE is read and parsed into a list of TLVs, which are store in `self.value`, by `_GeneralObject.read`. Then `read` is called on each member to process each TLV in turn. The next TLV is moved onto only when a member returns TRUE from the read call.

The `read` Method

Most of the logic for reading is implemented in `readContents` so it can be reused for `SequenceOf`'s `read` function.

The `write` Method

`self.value` is set to the contents of this SEQUENCE and then written by calling `_GeneralObject.write` whos encoder will call `write` of each element in the list of contents in turn.

The `set` Method

Accessing and setting values for ASN1 objects is a bit of a thorny issue. The problem stems from the arbitrary complexity of the data and the possible levels of nesting, which in practice are used and are quite massive. Designing a good general approach is a bit tricky, perhaps nearly impossible. I choose to use a most compact form which is excellent for simple objects and is very concise.

`value` should be a list or tuple of values. Each element of the list (or tuple) will be used in turn to set a member. Defaults can be specified by using the default value itself or `None`. Hence, for SEQUENCES of SEQUENCES, SEQUENCES OF, SET and so on `values` should consist of nested lists or tuples. Look at the ASN1 specs for that object to figure out exactly what these should look like.

The `get` Method

A tuple of the values of the contents of this sequence will be returned. Hence, for SEQUENCES of SEQUENCES, SEQUENCES OF, SET and so on nested tuples will be returned. `get` always returns tuples even if a list was used to set and object.

The `SequenceOf` Class

This object represents the ASN1 SEQUENCE OF construct.

Class Prototypes

```
class SequenceOf (Sequence):
    def __init__(contains, optional=0, default=”):
```

The `__init__` Method

The `contains` should be the constructor for the objects which this SEQUENCE OF contains.

The `Set` Class

This object represents the ASN1 Set type.

Class Prototypes

```
class Set (Sequence):
    def __init__(contents, optional=0, default=”):
```

The `__init__` Method

The `contents` should be a list containing the contents of the sequence.

The `SetOf` Class

This object represents the ASN1 SET OF construct.

Class Prototypes

```
class SetOf (SequenceOf):
    def __init__(contains, optional=0, default=”):
```

The `__init__` Method

The `contains` should be the constructor for the objects which this SET OF contains.

The `Explicit` Class

Explicit objects support the DER concept of explicit tagging. In general they behave just like a SEQUENCE which must have only one element. See below for other differences.

Class Prototypes

```
class Explicit (Sequence):
    def __init__(expclass, expform, expnumber, contents, optional=0, default=”):
    def set(value):
    def get():
```

The `__init__` Method

`expclass`, `expform`, `expnumber` should be as specified in the ASN1 documentation for this object. `contents` should be an object instance such as `Integer`, `Oid` or a derived object which supports the `_GeneralObjec` interface.

The `set` Method

`value` is passed direct to `set` of the explicit object, so it should not be placed in a list or tuple(unless you are setting a constructed object).

The `get` Method

The value of explicit object is returned and not put in a tuple.

The `Choice` Class

This object represents the ASN1 Choice type.

Class Prototypes

```
class Choice :
    def __init__(choices, optional=0, default=”):
    def reset():
    def set(value):
    def get():
    def toString():
    def fromString():
    def read(io):
    def write(file):
```

The `__init__` Method

`choices` should be a dictionary of objects which support the `_GeneralObject` interface. The key being the name of the choice specified in the ASN1 documentation. `optional` is a boolean indicating if this object is optional. The final parameter, `default` is the base 64 encoded DER value, which should be used as the default in leu of a value to read or incase it is unset. If neither `optional` or `default` is not set then the first choice which is optional or has a default will be honored.

The `reset` Method

This function re-initialises the object, clearing the value or setting it to any default.

The `set` Method

`value` should be a list or tuple with two elements. The first value should be the name of the choice to be set and the second the value to set it with.

The `get` Method

This function will return tuple with two elements. The first value will be the name of the choice which was set and the second the value it was set to.

The `toString` Method

Encodes the value in DER and returns it as a string.

The `fromString` Method

Decodes the string and sets the value of this object.

The `read` Method

`io` should be a file like object. If the object being read matches the expected class, form and tag the value is read and decoded using `decRoutine`. Else, if it has a default that is read and stored.

The return value of this function does not indicate success but whether this TLV was processed successfully. This bahaviour is vital for processing constructed types since the object may be optional or have a default. Failure to decode would be indicated by an exception.

The `write` Method

If this object has not been set and is not optional and dosn't have a default, a `DerError` exception will be raised

If no value has been set and this object is optional, nothing is written. If this object's value is equal to the default, nothing is written as stipulated by DER. Otherwise the value is encoded and written.

The `Certificate` Class

Example 2-1. Setting `Certificate`

```
rsa = POW.Asymmetric()
cert = POW.pkix.Certificate()
cert.setVersion(1)
cert.setSerial(5)

name = ( (( o2i('countryName'), ('printableString', 'GB') ),),
         (( o2i('stateOrProvinceName'), ('printableString', 'Hertfordshire') ),),
         (( o2i('organizationName'), ('printableString', 'The House') ),),
         (( o2i('commonName'), ('printableString', 'Client') ),) )

cert.setIssuer(name)
cert.setSubject(name)

now = POW.pkix.time2gen( time.time() )
then = POW.pkix.time2gen(time.time() + 60*60*24*365*12)
cert.setNotBefore( ('generalTime',  now) )
cert.setNotAfter( ( 'generalTime',  then) )
cert.setIssuerUniqueID((1,0,1,0))
cert.setSubjectUniqueID((1,0,0,1))
cert.sign(rsa, POW.MD5_DIGEST)
```

Class Prototypes

```
class Certificate (Sequence):
    def __init__(optional=0, default=”):
    def setVersion(version):
    def getVersion():
    def setSerial(serial):
    def getVersion():
    def setIssuer(names):
    def getIssuer():
    def setSubject(names):
    def getSubject():
    def setNotBefore(time):
    def getNotBefore():
    def setNotAfter(time):
    def getNotAfter():
    def setIssuerUniqueID(id):
    def getIssuerUniqueID():
    def setSubjectUniqueID(id):
    def getSubjectUniqueID():
    def setExtensions(extns):
    def getExtensions():
    def sign(rsa, digestType):
    def verify(rsa):
    def sign():
```

The `setVersion` Method

This function sets an `Integer` object. 0 indicates a version 1 certificate, 1 a version 2 certificate and 2 a version 3 certificate.

The `getVersion` Method

This function returns whatever the version object is set to, this should be 0, 1 or 2.

The `setSerial` Method

This function sets an `Integer` object. No two certificates issued should ever have the same serial number.

The `getVersion` Method

This function returns whatever the serial object is set to.

The `setIssuer` Method

This function sets an `Name` object. See `Certificate` class for an example.

The `getIssuer` Method

This function returns a complex tuple containing other tuples.

The `setSubject` Method

This function sets an `Name` object. See `Certificate` class for an example.

The `getSubject` Method

This function returns a complex tuple containing other tuples.

The `setNotBefore` Method

This function sets a `Choice` object. It can be either a `GeneralTime` or `UTCTime` object. The functions `gen2time`, `utc2time`, `time2gen` and `time2utc` can be used to convert to and from integer times and their string representation.

Example 2-2. `setNotBefore` method usage

```
cert = POW.pkix.Certificate()
now = POW.pkix.time2gen( time.time() )
cert.setNotBefore( ('generalTime',  now) )
```

The `getNotBefore` Method

This function returns a tuple indicating which type of time was stored and its value. See `setNotBefore` for details.

The `setNotAfter` Method

This function sets a `Choice` object. See `setNotBefore` for details.

The `getNotAfter` Method

This function returns a tuple indicating which type of time was stored and its value. See `setNotBefore` for details.

The `setIssuerUniqueID` Method

This function sets a `BitString` object. This is part of the X509v2 standard and is quite poorly regarded in general, its use is not recommended. It is set using the normal `BitString` method, that is with a sequence of true/false objects.

The `getIssuerUniqueID` Method

This function returns a tuple of integers, 1 or 0.

The `setSubjectUniqueID` Method

This function sets a `BitString` object. This is part of the X509v2 standard and is quite poorly regarded in general, its use is not recommended. It is set using the normal `BitString` method, that is with a sequence of true/false objects.

The `getSubjectUniqueID` Method

This function returns a tuple of integers, 1 or 0.

The `setExtensions` Method

This method sets an `Extensions` object, defined as SEQUENCE OF Extension. The parameter `extns` should consist of a list or tuple of values suitable to set an extension. See the extension class for details.

The `getExtensions` Method

This function returns a tuple of `Extension` values. See `Extension` for details.

The `sign` Method

This function updates structured of the `Certificate` and `tbs` as appropriate and performs the specified digest on the `tbs` and set `signedText` to signed the digest.

The `verify` Method

This function works out what kind of digest was used to during signing, calculates the digest of `tbs` and verifies the envelope using the key.

The `sign` Method

This function updates structured of the `certificateList` and `tBSCertList` as appropriate, performs the specified digest on the `tBSCertList` and sets `signedValue` to signed the digest.

The `CertificateList` Class

Example 2-3. Setting `CertificateList`

```
now = POW.pkix.time2gen( time.time() )
then = POW.pkix.time2gen(time.time() + 60*60*24*365*12)
rsa = POW.Asymmetric()

crl = POW.pkix.CertificateList()
crl.setThisUpdate( ('generalTime', now ) )

name = ( (( o2i('countryName'), ('printableString', 'GB') ),),
         (( o2i('stateOrProvinceName'), ('printableString', 'Hertfordshire') ),),
         (( o2i('organizationName'), ('printableString', 'The House') ),),
         (( o2i('commonName'), ('printableString', 'Client') ),) )

myRevocations = (
                  (1, ('generalTime', now), ()),
                  (2, ('generalTime', now), ()),
                  (3, ('generalTime', now), (( o2i('cRLReason'), 0, 1),))
                )

crl.setIssuer(name)
crl.setRevokedCertificates( myRevocations )

crl.sign(rsa, POW.MD5_DIGEST)
```

Class Prototypes

```
class CertificateList (Sequence):
    def __init__(optional=0, default=”):
    def setVersion(version):
    def getVersion():
    def setIssuer(names):
    def getIssuer():
    def getThisUpdate():
    def setNextUpdate():
    def getNextUpdate():
    def setExtensions(extns):
    def getExtensions():
    def setRevokedCertificates():
    def getRevokedCertificates():
    def verify():
```

The `setVersion` Method

This function sets an `Integer` object. 0 indicates a version 1 CRL, and 1 a version 2 CRL.

The `getVersion` Method

This function returns whatever the version object is set to, this should be 0, 1 or 2.

The `setIssuer` Method

This function sets an `Name` object.

The `getIssuer` Method

This function returns a complex tuple containing other tuples.

The `getThisUpdate` Method

This function returns a tuple containing two strings. The first is either ’utcTime’ or ’generalTime’ and the second is the time value as a string.

The `setNextUpdate` Method

See set `setThisUpdate`.

The `getNextUpdate` Method

See set `getThisUpdate`.

The `setExtensions` Method

This method sets an `Extensions` object, defined as SEQUENCE OF Extension. The parameter `extns` should consist of a list or tuple of values suitable to set an extension. See the extension class for details.

The `getExtensions` Method

This function returns a tuple of `Extension` values. See `Extension` for details.

The `setRevokedCertificates` Method

This function sets a sequence of `revokedCertificate` objects. This object is optional. See `CertificateList` for an example of its use.

The `getRevokedCertificates` Method

This function return a sequence of `revokedCertificate` objects or None.

The `verify` Method

This function works out what kind of digest was used to during signing, calculates the digest of `tBSCertList` and verifies the `signedText` using the key.

The `BasicConstraints` Class

This little extension has recently caused plenty of problems for several large organisations. It consist of a `Boolean` and an `Integer`. The first indicates if the owner is a CA, the second indicates how long a chain of CAs you should trust which the subject of this certificate trusts.

Example 2-4. Setting `BasicConstraints`

```
            bc = BasicConstraints()
            bc.set( (1, 1) )
```

Class Prototypes

```
class BasicConstraints (Sequence):
    def __init__(optional=0, default=”):
```

The `KeyUsage` Class

Class Prototypes

```
class KeyUsage (BitString):
```

The `SubjectAltName` Class

Class Prototypes

```
class SubjectAltName (GeneralNames):
```

The `IssuerAltName` Class

Class Prototypes

```
class IssuerAltName (GeneralNames):
```

The `SubjectKeyIdentifier` Class

Class Prototypes

```
class SubjectKeyIdentifier (OctetString):
```

The `AuthorityKeyIdentifier` Class

Example 2-5. Setting `AuthorityKeyIdentifier`

```
      id = AuthorityKeyIdentifier()
      authdigest = POW.Digest( POW.SHA1_DIGEST )
      authdigest.update(rsa.derWrite(POW.RSA_PUBLIC_KEY))
      keyHash = authdigest.digest()
      id.set( (keyHash, None, None) )
```

Class Prototypes

```
class AuthorityKeyIdentifier (Sequence):
    def __init__(optional=0, default=”):
```

The `PrivateKeyUsagePeriod` Class

Example 2-6. Setting `PrivateKeyUsagePeriod`

```
      period = PrivateKeyUsagePeriod()
      period.set( ( time2gen( time.time() ), None) )
```

Class Prototypes

```
class PrivateKeyUsagePeriod (Sequence):
    def __init__(optional=0, default=”):
```

The `CertificatePolicies` Class

Example 2-7. Setting `CertificatePolicies`

```
       data = (
                  ( o2i(’id-cti-ets-proofOfReceipt’), (
                     (o2i(’cps’),     (’cPSuri’, ’http://www.p-s.org.uk/policies/policy1
                     (o2i(’unotice’), (   ’userNotice’,
                                          ((’visibleString’, ’The House’),(1,2,3)),
                                             (’visibleString’, ’We guarentee nothing’)))
                  )),
                  ( o2i(’id-cti-ets-proofOfOrigin’), (
                     (o2i(’cps’), (’cPSuri’, ’http://www.p-s.org.uk/policies/policy2’)),
                  ))
               )
      policies = CertificatePolicies()
      policies.set( data )
```

Class Prototypes

```
class CertificatePolicies (SequenceOf):
    def __init__(optional=0, default=”):
```

The `CRLDistrobutionPoints` Class

Example 2-8. Setting `CRLDistrobutionPoints`

```
      n1 =  (’directoryName’,
               (  (( o2i(’countryName’), (’printableString’, ’UK’) ),),
                  (( o2i(’stateOrProvinceName’), (’printableString’, ’Herts’) ),),
                  (( o2i(’organizationName’), (’printableString’, ’The House’) ),),
                  (( o2i(’commonName’), (’printableString’, ’Shannon Works’) ),) ) )

      n2 = (’iPAddress’, POW.pkix.ip42oct(192,168,100,51))

      data = ( ( (’fullName’,(n1, n2)), (1,1,1,1,1), (n1,) ), )
      points = CRLDistrobutionPoints()
      points.set( data )
```

Class Prototypes

```
class CRLDistrobutionPoints (SequenceOf):
    def __init__(optional=0, default=”):
```

The `CrlNumber` Class

Class Prototypes

```
class CrlNumber (Integer):
```

The `DeltaCrlIndicator` Class

Class Prototypes

```
class DeltaCrlIndicator (Integer):
```

The `InvalidityDate` Class

Class Prototypes

```
class InvalidityDate (GeneralizedTime):
```

The `CrlReason` Class

Class Prototypes

```
class CrlReason (Enum):
```

The `Extension` Class

This class is a useful little object. It is set by passing three values: an oid, an integer(a boolean really) and a value. The boolean indicates if this extension is critical. The value is used to set the extension once it has been created. The oid is used to create the correct object which, to be fully supported it must be one of these:

```
basicConstraints
subjectAltName
issuerAltName
authorityKeyIdentifier
privateKeyUsagePeriod
certificatePolicies
cRLDistributionPoints
subjectKeyIdentifier
keyUsage
crlNumber
deltaCrlIndicator
invalidityDate
crlReason
```

Example 2-9. Setting `Extension`

```
extn = Extension()
email = ('rfc822Name', 'peter_shannon@yahoo.com')
extn.set( (obj2oid('subjectAltName'),1, (email,)) )
```

Class Prototypes

```
class Extension (Sequence):
    def set(values):
    def get():
```

The `set` Method

`values` should be a sequence of three values, the oid, critical marker and a value to set the extension. If an unknown oid is passed to this function it will raise an exception. `critical` is a boolean. `value` will be used to set the extension after it has been created.

The `get` Method

There are several ways this function might fail to decode an extension. Firstly if the extension was marked critical but if the oid cannot be mapped to a class or If a failure occurs decoding the `extnValue`, an exception will be raised. If a failure occurred and the extension was not marked critical it will return a tuple like this: `(oid, critical, ())`. If no failures occur a tuple will be returned, containg the oid, critical and extension values.