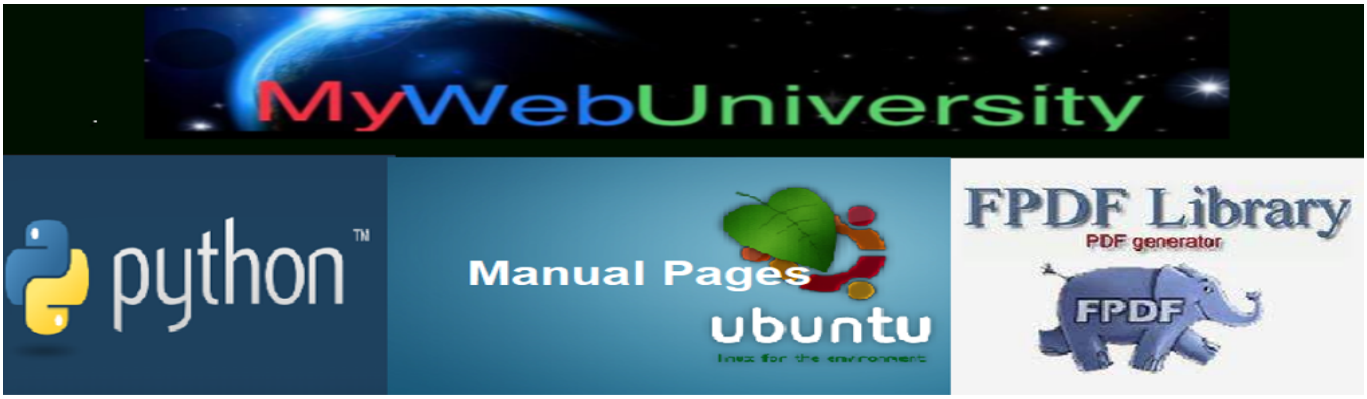

Rocky Enterprise Linux 9.2 Manual Pages on command 'passphrase-encoding.7ssl'

$ man passphrase-encoding.7ssl

PASSPHRASE-ENCODING(7SSL) OpenSSL PASSPHRASE-ENCODING(7SSL)

NAME

passphrase-encoding - How diverse parts of OpenSSL treat pass phrases character encoding

DESCRIPTION

In a modern world with all sorts of character encodings, the treatment of pass phrases has become increasingly complex. This manual page attempts to give an overview over how this problem is currently addressed in different parts of the OpenSSL library.

The general case

The OpenSSL library doesn't treat pass phrases in any special way as a general rule, and trusts the application or user to choose a suitable character set and stick to that throughout the lifetime of affected objects. This means that for an object that was encrypted using a pass phrase encoded in ISO-8859-1, that object needs to be decrypted using a pass phrase encoded in ISO-8859-1. Using the wrong encoding is expected to cause a decryption failure.

PKCS#12

PKCS#12 is a bit different regarding pass phrase encoding. The standard stipulates that the pass phrase shall be encoded as an ASN.1 BMPString, which consists of the code points of the basic multilingual plane, encoded in big endian (UCS-2 BE).

OpenSSL tries to adapt to this requirements in one of the following manners:

1. Treats the received pass phrase as UTF-8 encoded and tries to re-encode it to UTF-16 (which is the same as UCS-2 for characters U+0000 to U+D7FF and U+E000 to U+FFFF, but becomes an expansion for any other character), or failing that, proceeds with step 2.
2. Assumes that the pass phrase is encoded in ASCII or ISO-8859-1 and opportunistically

prepends each byte with a zero byte to obtain the UCS-2 encoding of the characters, which it stores as a BMPString.

Note that since there is no check of your locale, this may produce UCS-2 / UTF-16 characters that do not correspond to the original pass phrase characters for other character sets, such as any ISO-8859-X encoding other than ISO-8859-1 (or for Windows, CP 1252 with exception for the extra "graphical" characters in the 0x80-0x9F range).

OpenSSL versions older than 1.1.0 do variant 2 only, and that is the reason why OpenSSL still does this, to be able to read files produced with older versions.

It should be noted that this approach isn't entirely fault free.

A pass phrase encoded in ISO-8859-2 could very well have a sequence such as 0xC3 0xAF (which is the two characters "LATIN CAPITAL LETTER A WITH BREVE" and "LATIN CAPITAL LETTER Z WITH DOT ABOVE" in ISO-8859-2 encoding), but would be misinterpreted as the perfectly valid UTF-8 encoded code point U+00EF (LATIN SMALL LETTER I WITH DIAERESIS) if the pass phrase doesn't contain anything that would be invalid UTF-8. A pass phrase that contains this kind of byte sequence will give a different outcome in OpenSSL 1.1.0 and newer than in OpenSSL older than 1.1.0.

```
0x00 0xC3 0x00 0xAF             # OpenSSL older than 1.1.0
0x00 0xEF                       # OpenSSL 1.1.0 and newer
```

On the same accord, anything encoded in UTF-8 that was given to OpenSSL older than 1.1.0 was misinterpreted as ISO-8859-1 sequences.

OSSL_STORE

ossl_store(7) acts as a general interface to access all kinds of objects, potentially protected with a pass phrase, a PIN or something else. This API stipulates that pass phrases should be UTF-8 encoded, and that any other pass phrase encoding may give undefined results. This API relies on the application to ensure UTF-8 encoding, and doesn't check that this is the case, so what it gets, it will also pass to the underlying loader.

RECOMMENDATIONS

This section assumes that you know what pass phrase was used for encryption, but that it may have been encoded in a different character encoding than the one used by your current input method. For example, the pass phrase may have been used at a time when your default encoding was ISO-8859-1 (i.e. "naieve" resulting in the byte sequence 0x6E 0x61 0xEF 0x76 0x65), and you're now in an environment where your default encoding is UTF-8 (i.e.

"naieve" resulting in the byte sequence 0x6E 0x61 0xC3 0xAF 0x76 0x65). Whenever it's mentioned that you should use a certain character encoding, it should be understood that you either change the input method to use the mentioned encoding when you type in your pass phrase, or use some suitable tool to convert your pass phrase from your default encoding to the target encoding.

Also note that the sub-sections below discuss human readable pass phrases. This is particularly relevant for PKCS#12 objects, where human readable pass phrases are assumed. For other objects, it's as legitimate to use any byte sequence (such as a sequence of bytes from /dev/urandom that's been saved away), which makes any character encoding discussion irrelevant; in such cases, simply use the same byte sequence as it is.

Creating new objects

For creating new pass phrase protected objects, make sure the pass phrase is encoded using UTF-8. This is default on most modern Unixes, but may involve an effort on other platforms. Specifically for Windows, setting the environment variable OPENSSL_WIN32_UTF8 will have anything entered on [Windows] console prompt converted to UTF-8 (command line and separately prompted pass phrases alike).

Opening existing objects

For opening pass phrase protected objects where you know what character encoding was used for the encryption pass phrase, make sure to use the same encoding again.

For opening pass phrase protected objects where the character encoding that was used is unknown, or where the producing application is unknown, try one of the following:

1. Try the pass phrase that you have as it is in the character encoding of your environment. It's possible that its byte sequence is exactly right.
2. Convert the pass phrase to UTF-8 and try with the result. Specifically with PKCS#12, this should open up any object that was created according to the specification.
3. Do a naieve (i.e. purely mathematical) ISO-8859-1 to UTF-8 conversion and try with the result. This differs from the previous attempt because ISO-8859-1 maps directly to U+0000 to U+00FF, which other non-UTF-8 character sets do not.

   This also takes care of the case when a UTF-8 encoded string was used with OpenSSL older than 1.1.0. (for example, "ie", which is 0xC3 0xAF when encoded in UTF-8, would become 0xC3 0x83 0xC2 0xAF when re-encoded in the naieve manner. The conversion to BMPString would then yield 0x00 0xC3 0x00 0xA4 0x00 0x00, the erroneous/non-compliant encoding used by OpenSSL older than 1.1.0)

SEE ALSO

evp(7), ossl_store(7), EVP_BytesToKey(3), EVP_DecryptInit(3), PEM_do_header(3), PKCS12_parse(3), PKCS12_newpass(3), d2i_PKCS8PrivateKey_bio(3)

COPYRIGHT