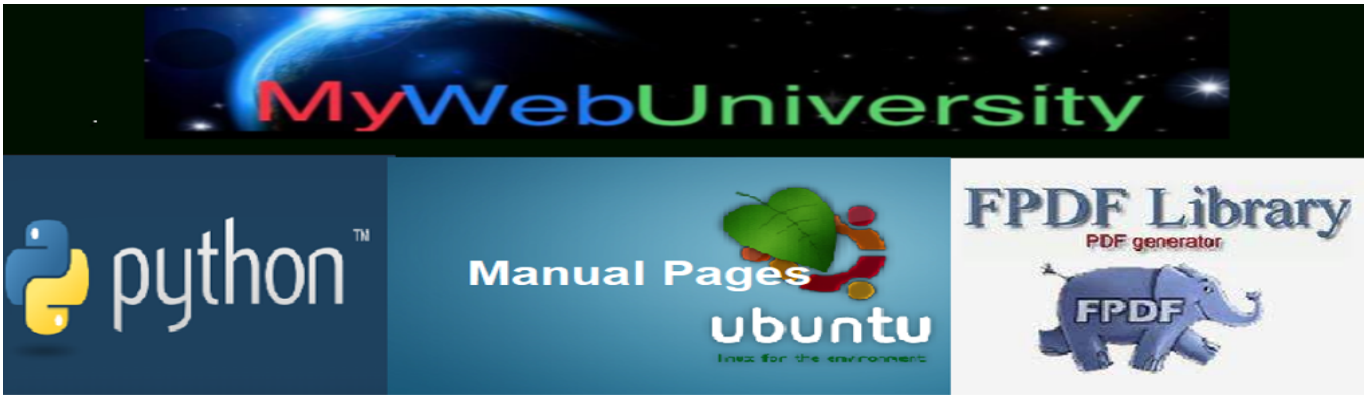

Rocky Enterprise Linux 9.2 Manual Pages on command 'EVP_KDF-SCRYPT.7ssl'

$ man EVP_KDF-SCRYPT.7ssl

NAME

EVP_KDF-SCRYPT - The scrypt EVP_KDF implementation

DESCRIPTION

Support for computing the scrypt password-based KDF through the EVP_KDF API.

The EVP_KDF-SCRYPT algorithm implements the scrypt password-based key derivation function, as described in RFC 7914. It is memory-hard in the sense that it deliberately requires a significant amount of RAM for efficient computation. The intention of this is to render brute forcing of passwords on systems that lack large amounts of main memory (such as GPUs or ASICs) computationally infeasible.

scrypt provides three work factors that can be customized: N, r and p. N, which has to be a positive power of two, is the general work factor and scales CPU time in an approximately linear fashion. r is the block size of the internally used hash function and p is the parallelization factor. Both r and p need to be greater than zero. The amount of RAM that scrypt requires for its computation is roughly (128 * N * r * p) bytes.

In the original paper of Colin Percival ("Stronger Key Derivation via Sequential Memory-Hard Functions", 2009), the suggested values that give a computation time of less than 5 seconds on a 2.5 GHz Intel Core 2 Duo are N = 2^20 = 1048576, r = 8, p = 1. Consequently,

the required amount of memory for this computation is roughly 1 GiB. On a more recent CPU (Intel i7-5930K at 3.5 GHz), this computation takes about 3 seconds. When N, r or p are not specified, they default to 1048576, 8, and 1, respectively. The maximum amount of RAM that may be used by scrypt defaults to 1025 MiB.

Identity

"SCRYPT" is the name for this implementation; it can be used with the EVP_KDF_fetch() function.

Supported parameters

The supported parameters are:

"pass" (OSSL_KDF_PARAM_PASSWORD) <octet string>

"salt" (OSSL_KDF_PARAM_SALT) <octet string>

These parameters work as described in "PARAMETERS" in EVP_KDF(3).

"n" (OSSL_KDF_PARAM_SCRYPT_N) <unsigned integer>

"r" (OSSL_KDF_PARAM_SCRYPT_R) <unsigned integer>

"p" (OSSL_KDF_PARAM_SCRYPT_P) <unsigned integer>

"maxmem_bytes" (OSSL_KDF_PARAM_SCRYPT_MAXMEM) <unsigned integer>

These parameters configure the scrypt work factors N, r, maxmem and p. Both N and maxmem_bytes are parameters of type uint64_t. Both r and p are parameters of type uint32_t.

"properties" (OSSL_KDF_PARAM_PROPERTIES) <UTF8 string>

This can be used to set the property query string when fetching the fixed digest internally. NULL is used if this value is not set.

NOTES

A context for scrypt can be obtained by calling:

```
EVP_KDF *kdf = EVP_KDF_fetch(NULL, "SCRYPT", NULL);
EVP_KDF_CTX *kctx = EVP_KDF_CTX_new(kdf);
```

The output length of an scrypt key derivation is specified via the "keylen" parameter to the EVP_KDF_derive(3) function.

EXAMPLES

This example derives a 64-byte long test vector using scrypt with the password "password", salt "NaCl" and N = 1024, r = 8, p = 16.

```
EVP_KDF *kdf;
EVP_KDF_CTX *kctx;
unsigned char out[64];
OSSL_PARAM params[6], *p = params;

kdf = EVP_KDF_fetch(NULL, "SCRYPT", NULL);
kctx = EVP_KDF_CTX_new(kdf);
EVP_KDF_free(kdf);

*p++ = OSSL_PARAM_construct_octet_string(OSSL_KDF_PARAM_PASSWORD,
                                         "password", (size_t)8);
*p++ = OSSL_PARAM_construct_octet_string(OSSL_KDF_PARAM_SALT,
                                         "NaCl", (size_t)4);
*p++ = OSSL_PARAM_construct_uint64(OSSL_KDF_PARAM_SCRYPT_N, (uint64_t)1024);
*p++ = OSSL_PARAM_construct_uint32(OSSL_KDF_PARAM_SCRYPT_R, (uint32_t)8);
*p++ = OSSL_PARAM_construct_uint32(OSSL_KDF_PARAM_SCRYPT_P, (uint32_t)16);
*p = OSSL_PARAM_construct_end();
if (EVP_KDF_derive(kctx, out, sizeof(out), params) <= 0) {
    error("EVP_KDF_derive");
}

{
    const unsigned char expected[sizeof(out)] = {
        0xfd, 0xba, 0xbe, 0x1c, 0x9d, 0x34, 0x72, 0x00,
        0x78, 0x56, 0xe7, 0x19, 0x0d, 0x01, 0xe9, 0xfe,
```

```
            0x7c, 0x6a, 0xd7, 0xcb, 0xc8, 0x23, 0x78, 0x30,
            0xe7, 0x73, 0x76, 0x63, 0x4b, 0x37, 0x31, 0x62,
            0x2e, 0xaf, 0x30, 0xd9, 0x2e, 0x22, 0xa3, 0x88,
            0x6f, 0xf1, 0x09, 0x27, 0x9d, 0x98, 0x30, 0xda,
            0xc7, 0x27, 0xaf, 0xb9, 0x4a, 0x83, 0xee, 0x6d,
            0x83, 0x60, 0xcb, 0xdf, 0xa2, 0xcc, 0x06, 0x40
        };

        assert(!memcmp(out, expected, sizeof(out)));
    }

    EVP_KDF_CTX_free(kctx);
```

CONFORMING TO

RFC 7914

SEE ALSO

EVP_KDF(3), EVP_KDF_CTX_new(3), EVP_KDF_CTX_free(3), EVP_KDF_CTX_set_params(3), EVP_KDF_derive(3), "PARAMETERS" in EVP_KDF(3)

COPYRIGHT