

## NAME

script – EVP\_PKEY script KDF support

## DESCRIPTION

The EVP\_PKEY\_SCRIPT algorithm implements the script 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.

script 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 script 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 default amount of RAM that may be used by script defaults to 1025 MiB.

## NOTES

A context for script can be obtained by calling:

```
EVP_PKEY_CTX *pctx = EVP_PKEY_CTX_new_id(EVP_PKEY_SCRIPT, NULL);
```

The output length of an script key derivation is specified via the length parameter to the **EVP\_PKEY\_derive**(3) function.

## EXAMPLE

This example derives a 64-byte long test vector using script using the password “password”, salt “NaCl” and  $N = 1024$ ,  $r = 8$ ,  $p = 16$ .

```
EVP_PKEY_CTX *pctx;
unsigned char out[64];

size_t outlen = sizeof(out);
pctx = EVP_PKEY_CTX_new_id(EVP_PKEY_SCRIPT, NULL);

if (EVP_PKEY_derive_init(pctx) <= 0) {
    error("EVP_PKEY_derive_init");
}
if (EVP_PKEY_CTX_set1_pbe_pass(pctx, "password", 8) <= 0) {
    error("EVP_PKEY_CTX_set1_pbe_pass");
}
if (EVP_PKEY_CTX_set1_script_salt(pctx, "NaCl", 4) <= 0) {
    error("EVP_PKEY_CTX_set1_script_salt");
}
if (EVP_PKEY_CTX_set_script_N(pctx, 1024) <= 0) {
    error("EVP_PKEY_CTX_set_script_N");
}
if (EVP_PKEY_CTX_set_script_r(pctx, 8) <= 0) {
    error("EVP_PKEY_CTX_set_script_r");
}
if (EVP_PKEY_CTX_set_script_p(pctx, 16) <= 0) {
    error("EVP_PKEY_CTX_set_script_p");
}
if (EVP_PKEY_derive(pctx, out, &outlen) <= 0) {
    error("EVP_PKEY_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_PKEY_CTX_free(pctx);

```

## CONFORMING TO

RFC 7914

## SEE ALSO

<b>EVP_PKEY_CTX_set1_scrypt_salt</b> (3),	<b>EVP_PKEY_CTX_set_scrypt_N</b> (3),
<b>EVP_PKEY_CTX_set_scrypt_r</b> (3),	<b>EVP_PKEY_CTX_set_scrypt_p</b> (3),
<b>EVP_PKEY_CTX_set_scrypt_maxmem_bytes</b> (3),	<b>EVP_PKEY_CTX_new</b> (3),
<b>EVP_PKEY_CTX_ctrl_str</b> (3), <b>EVP_PKEY_derive</b> (3)	

## COPYRIGHT

Copyright 2017–2018 The OpenSSL Project Authors. All Rights Reserved.

Licensed under the OpenSSL license (the “License”). You may not use this file except in compliance with the License. You can obtain a copy in the file LICENSE in the source distribution or at <https://www.openssl.org/source/license.html>.