Network Working Group C. Percival
Internet-Draft Tarsnap
Intended status: Informational S. Josefsson
Expires: July 30, 2015 SJD AB
January 26, 2015

The scrypt Password-Based Key Derivation Function
draft-josefsson-scrypt-kdf-02

Abstract

This document specifies the password-based key derivation function scrypt. The function derives one or more secret keys from a secret string. It is based on memory-hard functions which offer added protection against attacks using custom hardware. The document also provides an ASN.1 schema.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 30, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
2. The Salsa20/8 Core Function 3
3. The scryptBlockMix Algorithm 3
4. The scryptROMix Algorithm 4
5. The scrypt Algorithm 5
6. ASN.1 Syntax 6
   6.1. ASN.1 Module 7
7. Test Vectors for Salsa20/8 Core 8
8. Test Vectors for scryptBlockMix 8
9. Test Vectors for scryptROMix 9
10. Test Vectors for PBKDF2 with HMAC-SHA-256 9
11. Test Vectors for scrypt 10
12. Copying Conditions 11
13. Acknowledgements 11
14. IANA Considerations 11
15. Security Considerations 11
16. References 11
   16.1. Normative References 12
   16.2. Informative References 12
Authors' Addresses 12

1. Introduction

Password-based key derivation functions are used in cryptography for deriving one or more secret keys from a secret value. Over the years, several password-based key derivation functions have been used, including the original DES-based UNIX Crypt-function, FreeBSD MD5 crypt, PKCS#5 PBKDF2 [RFC2898] (typically used with SHA-1), GNU SHA-256/512 crypt, Windows NT LAN Manager (NTLM) hash, and the Blowfish-based bcrypt. These algorithms are based on similar techniques that employ a cryptographic primitive, salting and/or iteration. The iteration count is used to slow down the computation.

Providing that the number of iterations used is increased as computer systems get faster, this allows legitimate users to spend a constant amount of time on key derivation without losing ground to an attackers' ever-increasing computing power - as long as attackers are limited to the same software implementations as legitimate users. However, as Bernstein pointed out in the context of integer factorization, while parallelized hardware implementations may not change the number of operations performed compared to software implementations, this does not prevent them from dramatically changing the asymptotic cost, since in many contexts - including the

embarrassingly parallel task of performing a brute-force search for a passphrase - dollar-seconds are the most appropriate units for measuring the cost of a computation. As semiconductor technology develops, circuits do not merely become faster; they also become smaller, allowing for a larger amount of parallelism at the same cost. Consequently, existing key derivation algorithms, even when the iteration count is increased so that the time taken to verify a password remains constant, the cost of finding a password by using a brute force attack implemented in hardware drops each year.

The scrypt function aims to reduce the advantage which attackers can gain by using custom-designed parallel circuits for breaking password-based key derivation functions.

For further background, see the original scrypt paper [SCRYPT].

The rest of this document is divided into sections that each describe algorithms needed for the final "scrypt" algorithm.

2. The Salsa20/8 Core Function

Salsa20/8 Core is a round-reduced variant of the Salsa20 Core. It is a hash function from 64-octet strings to 64-octet strings. Note that Salsa20/8 Core is not a cryptographic hash function since it is not collision-resistant. See section 8 of [SALSA20SPEC] for its specification, and [SALSA20CORE] for more information.

3. The scryptBlockMix Algorithm

The scryptBlockMix algorithm is the same as the BlockMix algorithm described in [SCRYPT] but with Salsa20/8 Core used as the hash function H. Below, Salsa(T) corresponds to the Salsa20/8 Core function applied to the octet vector T.

```
Algorithm scryptBlockMix

Parameters:
         r       Block size parameter.

Input:
         B[0], ..., B[2 * r - 1]
                 Input vector of 2 * r 64-octet blocks.

Output:
         B'[0], ..., B'[2 * r - 1]
                 Output vector of 2 * r 64-octet blocks.

Steps:

  1. X = B[2 * r - 1]

  2. for i = 0 to 2 * r - 1 do
       T = X xor B[i]
       X = Salsa (T)
       Y[i] = X
     end for

  3. B' = (Y[0], Y[2], ..., Y[2 * r - 2],
           Y[1], Y[3], ..., Y[2 * r - 1])
```

4. The scryptROMix Algorithm

The scryptROMix algorithm is the same as the ROMix algorithm described in [SCRYPT] but with scryptBlockMix used as the hash function H and the Integerify function explained inline.

```
Algorithm scryptROMix

Input:
         r       Block size parameter.
         B       Input octet vector of length 128 * r octets.
         N       CPU/Memory cost parameter, must be larger than 1,
                 a power of 2 and less than 2^(128 * r / 8).

Output:
         B'      Output octet vector of length 128 * r octets.

Steps:

  1. X = B

  2. for i = 0 to N - 1 do
       V[i] = X
       X = scryptBlockMix (X)
     end for

  3. for i = 0 to N - 1 do
       j = Integerify (X) mod N
              where Integerify (B[0] ... B[2 * r - 1]) is defined
              as the result of interpreting B[2 * r - 1] as a
              little-endian integer.
       T = X xor V[j]
       X = scryptBlockMix (T)
     end for

  4. B' = X
```

5. The scrypt Algorithm

The PBKDF2-HMAC-SHA-256 function used below denote the PBKDF2 algorithm [RFC2898] used with HMAC-SHA-256 [RFC6234] as the PRF. The HMAC-SHA-256 function generates 32 octet outputs.

```
Algorithm scrypt

Input:
         P       Passphrase, an octet string.
         S       Salt, an octet string.
         N       CPU/Memory cost parameter, must be larger than 1,
                 a power of 2 and less than 2^(128 * r / 8).
         r       Block size parameter.
         p       Parallelization parameter, a positive integer
                 less than or equal to ((2^32-1) * hLen) / MFLen
                 where hLen is 32 and MFlen is 128 * r.
         dkLen   Intended output length in octets of the derived
                 key; a positive integer less than or equal to
                 (2^32 - 1) * hLen where hLen is 32.

Output:
         DK      Derived key, of length dkLen octets.

Steps:

  1. B[0] || B[1] || ... || B[p - 1] =
       PBKDF2-HMAC-SHA256 (P, S, 1, p * 128 * r)

  2. for i = 0 to p - 1 do
       B[i] = scryptROMix (r, B[i], N)
     end for

  3. DK = PBKDF2-HMAC-SHA256 (P, B[0] || B[1] || ... || B[p - 1],
                              1, dkLen)
```

6. ASN.1 Syntax

This section defines ASN.1 syntax for the scrypt key derivation function. The intended application of these definitions includes PKCS #8 and other syntax for key management. (Various aspects of ASN.1 are specified in several ISO/IEC standards.)

The object identifier id-scrypt identifies the scrypt key derivation function.

```
id-scrypt OBJECT IDENTIFIER ::= {1 3 6 1 4 1 11591 4 11}
```

The parameters field associated with this OID in an AlgorithmIdentifier shall have type scrypt-params:

```
scrypt-params ::= SEQUENCE {
       salt OCTET STRING,
       costParameter INTEGER (1..MAX),
       blockSize INTEGER (1..MAX),
       parallelizationParameter INTEGER (1..MAX),
       keyLength INTEGER (1..MAX) OPTIONAL }
```

The fields of type scrypt-params have the following meanings:

- salt specifies the salt value. It shall be an octet string.

- costParameter specifies the CPU/Memory cost parameter N.

- blockSize specifies the block size parameter r.

- parallelizationParameter specifies the parallelization parameter.

- keyLength, an optional field, is the length in octets of the derived key. The maximum key length allowed depends on the implementation; it is expected that implementation profiles may further constrain the bounds. This field only provides convenience; the key length is not cryptographically protected.

6.1. ASN.1 Module

For reference purposes, the ASN.1 syntax is presented as an ASN.1 module here.

```
-- scrypt ASN.1 Module

scrypt-0 {1 3 6 1 4 1 11591 4 10}

DEFINITIONS ::= BEGIN

id-scrypt OBJECT IDENTIFIER ::= {1 3 6 1 4 1 11591 4 11}

scrypt-params ::= SEQUENCE {
    salt OCTET STRING,
    costParameter INTEGER (1..MAX),
    blockSize INTEGER (1..MAX),
    parallelizationParameter INTEGER (1..MAX),
    keyLength INTEGER (1..MAX) OPTIONAL
}

END
```

7. Test Vectors for Salsa20/8 Core

Below is a sequence of octets to illustrate input and output values for the Salsa20/8 Core. The octets are hex encoded and whitespace is inserted for readability. The value corresponds to the first input and output pair generated by the first scrypt test vector below.

```
INPUT:
7e 87 9a 21 4f 3e c9 86 7c a9 40 e6 41 71 8f 26
ba ee 55 5b 8c 61 c1 b5 0d f8 46 11 6d cd 3b 1d
ee 24 f3 19 df 9b 3d 85 14 12 1e 4b 5a c5 aa 32
76 02 1d 29 09 c7 48 29 ed eb c6 8d b8 b8 c2 5e

OUTPUT:
a4 1f 85 9c 66 08 cc 99 3b 81 ca cb 02 0c ef 05
04 4b 21 81 a2 fd 33 7d fd 7b 1c 63 96 68 2f 29
b4 39 31 68 e3 c9 e6 bc fe 6b c5 b7 a0 6d 96 ba
e4 24 cc 10 2c 91 74 5c 24 ad 67 3d c7 61 8f 81
```

8. Test Vectors for scryptBlockMix

Below is a sequence of octets to illustrate input and output values for scryptBlockMix. The test vector uses an r value of 1. The octets are hex encoded and whitespace is inserted for readability. The value corresponds to the first input and output pair generated by the first scrypt test vector below.

```
INPUT
B[0] =  f7 ce 0b 65 3d 2d 72 a4 10 8c f5 ab e9 12 ff dd
        77 76 16 db bb 27 a7 0e 82 04 f3 ae 2d 0f 6f ad
        89 f6 8f 48 11 d1 e8 7b cc 3b d7 40 0a 9f fd 29
        09 4f 01 84 63 95 74 f3 9a e5 a1 31 52 17 bc d7

B[1] =  89 49 91 44 72 13 bb 22 6c 25 b5 4d a8 63 70 fb
        cd 98 43 80 37 46 66 bb 8f fc b5 bf 40 c2 54 b0
        67 d2 7c 51 ce 4a d5 fe d8 29 c9 0b 50 5a 57 1b
        7f 4d 1c ad 6a 52 3c da 77 0e 67 bc ea af 7e 89

OUTPUT
B'[0] = a4 1f 85 9c 66 08 cc 99 3b 81 ca cb 02 0c ef 05
        04 4b 21 81 a2 fd 33 7d fd 7b 1c 63 96 68 2f 29
        b4 39 31 68 e3 c9 e6 bc fe 6b c5 b7 a0 6d 96 ba
        e4 24 cc 10 2c 91 74 5c 24 ad 67 3d c7 61 8f 81

B'[1] = 20 ed c9 75 32 38 81 a8 05 40 f6 4c 16 2d cd 3c
        21 07 7c fe 5f 8d 5f e2 b1 a4 16 8f 95 36 78 b7
        7d 3b 3d 80 3b 60 e4 ab 92 09 96 e5 9b 4d 53 b6
        5d 2a 22 58 77 d5 ed f5 84 2c b9 f1 4e ef e4 25
```

9. Test Vectors for scryptROMix

Below is a sequence of octets to illustrate input and output values for scryptROMix. The test vector uses an r value of 1 and an N value of 16. The octets are hex encoded and whitespace is inserted for readability. The value corresponds to the first input and output pair generated by the first scrypt test vector below.

```
INPUT:
B = f7 ce 0b 65 3d 2d 72 a4 10 8c f5 ab e9 12 ff dd
    77 76 16 db bb 27 a7 0e 82 04 f3 ae 2d 0f 6f ad
    89 f6 8f 48 11 d1 e8 7b cc 3b d7 40 0a 9f fd 29
    09 4f 01 84 63 95 74 f3 9a e5 a1 31 52 17 bc d7
    89 49 91 44 72 13 bb 22 6c 25 b5 4d a8 63 70 fb
    cd 98 43 80 37 46 66 bb 8f fc b5 bf 40 c2 54 b0
    67 d2 7c 51 ce 4a d5 fe d8 29 c9 0b 50 5a 57 1b
    7f 4d 1c ad 6a 52 3c da 77 0e 67 bc ea af 7e 89

OUTPUT:
B = 79 cc c1 93 62 9d eb ca 04 7f 0b 70 60 4b f6 b6
    2c e3 dd 4a 96 26 e3 55 fa fc 61 98 e6 ea 2b 46
    d5 84 13 67 3b 99 b0 29 d6 65 c3 57 60 1f b4 26
    a0 b2 f4 bb a2 00 ee 9f 0a 43 d1 9b 57 1a 9c 71
    ef 11 42 e6 5d 5a 26 6f dd ca 83 2c e5 9f aa 7c
    ac 0b 9c f1 be 2b ff ca 30 0d 01 ee 38 76 19 c4
    ae 12 fd 44 38 f2 03 a0 e4 e1 c4 7e c3 14 86 1f
    4e 90 87 cb 33 39 6a 68 73 e8 f9 d2 53 9a 4b 8e
```

10. Test Vectors for PBKDF2 with HMAC-SHA-256

Below is a sequence of octets illustring input and output values for PBKDF2-HMAC-SHA-256. The octets are hex encoded and whitespace is inserted for readability. The test vectors below can be used to verify the PBKDF2-HMAC-SHA-256 [RFC2898] function. The password and salt strings are passed as sequences of ASCII [RFC0020] octets.

```
PBKDF2-HMAC-SHA-256 (P="passwd", S="salt",
                    c=1, dkLen=64) =
55 ac 04 6e 56 e3 08 9f ec 16 91 c2 25 44 b6 05
f9 41 85 21 6d de 04 65 e6 8b 9d 57 c2 0d ac bc
49 ca 9c cc f1 79 b6 45 99 16 64 b3 9d 77 ef 31
7c 71 b8 45 b1 e3 0b d5 09 11 20 41 d3 a1 97 83
```

```
PBKDF2-HMAC-SHA-256 (P="Password", S="NaCl",
                     c=80000, dkLen=64) =
4d dc d8 f6 0b 98 be 21 83 0c ee 5e f2 27 01 f9
64 1a 44 18 d0 4c 04 14 ae ff 08 87 6b 34 ab 56
a1 d4 25 a1 22 58 33 54 9a db 84 1b 51 c9 b3 17
6a 27 2b de bb a1 d0 78 47 8f 62 b3 97 f3 3c 8d
```

11. Test Vectors for scrypt

For reference purposes, we provide the following test vectors for scrypt, where the password and salt strings are passed as sequences of ASCII [RFC0020] octets.

The parameters to the scrypt function below are, in order, the password P (octet string), the salt S (octet string), the CPU/Memory cost parameter N, the block size parameter r, and the parallelization parameter p, and the output size dkLen. The output is hex encoded and whitespace is inserted for readability.

```
scrypt (P="", S="",
        N=16, r=1, p=1, dklen=64) =
77 d6 57 62 38 65 7b 20 3b 19 ca 42 c1 8a 04 97
f1 6b 48 44 e3 07 4a e8 df df fa 3f ed e2 14 42
fc d0 06 9d ed 09 48 f8 32 6a 75 3a 0f c8 1f 17
e8 d3 e0 fb 2e 0d 36 28 cf 35 e2 0c 38 d1 89 06

scrypt (P="password", S="NaCl",
        N=1024, r=8, p=16, dkLen=64) =
fd ba be 1c 9d 34 72 00 78 56 e7 19 0d 01 e9 fe
7c 6a d7 cb c8 23 78 30 e7 73 76 63 4b 37 31 62
2e af 30 d9 2e 22 a3 88 6f f1 09 27 9d 98 30 da
c7 27 af b9 4a 83 ee 6d 83 60 cb df a2 cc 06 40

scrypt (P="pleaseletmein", S="SodiumChloride",
        N=16384, r=8, p=1, dkLen=64) =
70 23 bd cb 3a fd 73 48 46 1c 06 cd 81 fd 38 eb
fd a8 fb ba 90 4f 8e 3e a9 b5 43 f6 54 5d a1 f2
d5 43 29 55 61 3f 0f cf 62 d4 97 05 24 2a 9a f9
e6 1e 85 dc 0d 65 1e 40 df cf 01 7b 45 57 58 87

scrypt (P="pleaseletmein", S="SodiumChloride",
        N=1048576, r=8, p=1, dkLen=64) =
21 01 cb 9b 6a 51 1a ae ad db be 09 cf 70 f8 81
ec 56 8d 57 4a 2f fd 4d ab e5 ee 98 20 ad aa 47
8e 56 fd 8f 4b a5 d0 9f fa 1c 6d 92 7c 40 f4 c3
37 30 40 49 e8 a9 52 fb cb f4 5c 6f a7 7a 41 a4
```

12. Copying Conditions

The authors agree to grant third parties the irrevocable right to copy, use and distribute this entire document or any portion of it, with or without modification, in any medium, without royalty, provided that, unless separate permission is granted, redistributed modified works do not contain misleading author, version, name of work, or endorsement information.

13. Acknowledgements

Text in this document was borrowed from [SCRYPT] and [RFC2898].

Feedback on this document were received from Dmitry Chestnykh, Alexander Klink, Rob Kendrick, Royce Williams Ted Rolle, Jr., and Eitan Adler.

14. IANA Considerations

None.

15. Security Considerations

This document specifies a cryptographic algorithm. The reader must follow cryptographic research of published attacks. ROMix has been proven sequential memory-hard under the Random Oracle model for the hash function. The security of scrypt relies on the assumption that BlockMix with Salsa20/8 Core does not exhibit any "shortcuts" which would allow it to be iterated more easily than a random oracle. For other claims about the security properties see [SCRYPT].

Passwords and other sensitive data, such as intermediate values, may continue to be stored in memory, core dumps, swap areas, etc, for a long time after the implementation has processed them. This makes attacks on the implementation easier. Thus, implementation should consider storing sensitive data in protected memory areas. How to achieve this is system dependent.

By nature and depending on parameters, running the scrypt algorithm may require large amounts of memory. Systems should protect against a denial of service attack resulting from attackers presenting unreasonably large parameters.

16. References

16.1. Normative References

[RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, September 2000.

[RFC6234] Eastlake, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, May 2011.

[SALSA20SPEC] Bernstein, D., "Salsa20 specification", WWW http://cr.yp.to/snuffle/spec.pdf, April 2005.

[SALSA20CORE] Bernstein, D., "The Salsa20 Core", WWW http://cr.yp.to/salsa20.html, March 2005.

16.2. Informative References

[RFC0020] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

[SCRYPT] Percival, C., "Stronger key derivation via sequential memory-hard functions", BSDCan'09 http://www.tarsnap.com/scrypt/scrypt.pdf, May 2009.

Authors' Addresses

Colin Percival
Tarsnap

Email: cperciva@tarsnap.com

Simon Josefsson
SJD AB

Email: simon@josefsson.org
URI: http://josefsson.org/