Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 8, 2016

A. Biryukov
D. Dinu
D. Khovratovich
University of Luxembourg
S. Josefsson
SJD AB

November 5, 2015

The memory-hard Argon2 password hash function
draft-josefsson-argon2-00

Abstract

This document describes the Argon2 memory-hard function for password hashing and other applications. We provide a implementer oriented description together with sample code and test vectors. The purpose is to simplify adoption of Argon2 for Internet protocols.

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 May 8, 2016.

Copyright Notice

Table of Contents

1. Introduction

This document describes the Argon2 memory-hard function for password hashing and other applications. We provide a implementer oriented description together with sample code and test vectors. The purpose is to simplify adoption of Argon2 for Internet protocols.

Argon2 summarizes the state of the art in the design of memory-hard functions. It is a streamlined and simple design. It aims at the highest memory filling rate and effective use of multiple computing units, while still providing defense against tradeoff attacks. Argon2 is optimized for the x86 architecture and exploits the cache and memory organization of the recent Intel and AMD processors. Argon2 has two variants: Argon2d and Argon2i. Argon2d is faster and uses data-depending memory access, which makes it suitable for cryptocurrencies and applications with no threats from side-channel timing attacks. Argon2i uses data-independent memory access, which is preferred for password hashing and password-based key derivation. Argon2i is slower as it makes more passes over the memory to protect from tradeoff attacks.

For further background and discussion, see the Argon2 paper [ARGON2].

2. Notation and Conventions

x^y --- x multiplied by itself y times

a*b --- multiplication of a and b

c-d --- substraction of c with d

E_f --- variable E with subscript index f

g / h --- g divided by h

I(j) --- function I evaluated on parameters j

K || L --- string K concatenated with string L

3. Argon2 Algorithm

3.1. Argon2 Inputs and Outputs

Argon2 have the following input parameters:

- Message string P, typically a password. May have any length from 0 to 2^32 - 1 bytes.
- Nonce S, typically a random salt. May have any length from 8 to 2^32 - 1 bytes. 16 bytes is recommended for password hashing. See [RFC4086] for discussion about randomness.
- Degree of parallelism p determines how many independent (but synchronizing) computational chains can be run. It may take any integer value from 1 to 255.
- Tag length T may be any integer number of bytes from 4 to 2^32-1.
- Memory size m can be any integer number of kilobytes from 8*p to 2^32-1. The actual number of blocks is m', which is m rounded down to the nearest multiple of 4*p.
- Number of iterations t (used to tune the running time independently of the memory size) can be any integer number from 1 to 2^32-1.
- Version number v is one byte 0x10.

- Secret value K (serves as key if necessary, but we do not assume any key use by default) may have any length from 0 to 32 bytes.

- Associated data X may have any length from 0 to 2^32-1 bytes.

- Type y of Argon2: 0 for Argon2d, 1 for Argon2i.

The Argon2 output is a T-length string.

3.2. Argon2 Operation

Argon2 uses an internal compression function G with two 1024-byte inputs and a 1024-byte output, and an internal hash function H. Here H is the Blake2b [I-D.saarinen-blake2] hash function, and the compression function G is based on its internal permutation. A variable-length hash function H' built upon H is also used. G and H' are described in later section.

The Argon2 operation is as follows.

1. Establish H_0 as the 64-bit value as shown in the figure below. H is BLAKE2b and the non-strings p, T, m, t, v, y, length(P), length(S), length(K), and length(X) are treated as a 32-bit little-endian encoding of the integer.

```
H_0 = H(p, T, m, t, v, y, length(P), P, length(S), S,
        length(K), K, length(X), X)
```

2. Allocate the memory as m' 1024-byte blocks where m' is derived as:

```
m' = 4 * p * floor (m / 4p)
```

   For tunable parallelism with p threads, the memory is organized in a matrix B[i][j] of blocks with p rows (lanes) and q = m' / p columns.

3. Compute B[i][0] for all i ranging from (and including) 0 to (not including) p.

```
B[i][0] = H'(H0, 4byteencode(i), 4byteencode(0))
```

   Here 4byteencode is a function which takes an integer and little-endian encode and padds it to 4 bytes.

4. Compute B[i][1] for all i ranging from (and including) 0 to (not including) p.

```
B[i][1] = H'(H0, 4byteencode(i), 4byteencode(1))
```

5. Compute B[i][j] for all i ranging from (and including) 0 to (not including) p, and for all j ranging from (and including) 2) to (not including) q. The block indices i' and j' are determined differently for Argon2d and Argon2i.

```
B[i][j] = G(B[i][j-1], B[i'][j'])
```

6. If the number of iterations t is larger than 1, we repeat the steps however replacing the computations with with the following expression:

```
B[i][0] = G(B[i][q-1], B[i'][j'])
B[i][j] = G(B[i][j-1], B[i'][j'])
```

7. After t steps have been iterated, we compute the final block C as the XOR of the last column:

```
C = B[0][q-1] XOR B[1][q-1] XOR ... XOR B[p-1][q-1]
```

8. The output tag is computed as H'(C).

3.3. Variable-length hash function H'

Let H_x be a hash function with x-byte output (in our case H_x is Blake2b, which supports x between 1 and 64 inclusive). Let V_i be a 64-byte block, and A_i be its first 32 bytes, and T < 2^32 be the tag length in bytes. Then we define

```
V_0 = T||X
V_1 = H_64(V_0)
    V_2 = H_64(V_1)
...
V_r = H_64(V_{r-1})   with r=floor(T/32)-1
V_{r+1} = H_{T mod 64}(V_{r-1}) absent if 64 divides T
H'(X) = A_1 || A_2 || ... || A_r || V_{r+1}
```

FIXME: improve this description. FIXME2: V_{r+1} is not properly described, is it a 64-byte block or a {T mod 64} block?

3.4. Indexing

TBD

3.5. Compression function G

Compression function G is built upon the Blake2b round function P. P operates on the 128-byte input, which can be viewed as 8 16-byte registers:

P(A_0, A_1, ... ,A_7) = (B_0, B_1, ... ,B_7)

Compression function G(X, Y) operates on two 1024-byte blocks X and Y. It first computes R = X XOR Y. Then R is viewed as a 8x8-matrix of 16-byte registers R_0, R_1, ... , R_63. Then P is first applied rowwise, and then columnwise to get Z:

```
(Q_0, Q_1, ... , Q_7 ) <- P(R_0, R_1, ... , R_7)
(Q_8, Q_9, ... , Q_15) <- P(R_8, R_9, ... , R_15)
...
(Q_56, Q_57, ... , Q_63 ) <- P(R_56, R_57, ... , R_63)
(Z_0, Z_8, Z_16 , ... , Z_56) < P(Q_0, Q_8, Q_16, ... , Q_56)
(Z_1, Z_9, Z_17 , ... , Z_57) < P(Q_1, Q_9, Q_17, ... , Q_57)
...
(Z_7, Z_15, Z 23 , ... , Z_63) < P(Q_7, Q_15, Q_23, ... , Q_63)
```

Finally, G outputs Z XOR R:

G: (X,Y) -> R = X XOR Y -P-> Q -P-> Z -P-> Z XOR R

FIXME: improve this description.

3.6. Permutation P

TBD

4. Parameter Choice

Argon2d is optimized for settings where the adversary does not get regular access to system memory or CPU, i.e. he can not run side-channel attacks based on the timing information, nor he can recover the password much faster using garbage collection. These settings are more typical for backend servers and cryptocurrency minings. For practice we suggest the following settings:

- Cryptocurrency mining, that takes 0.1 seconds on a 2 Ghz CPU using 1 core -- Argon2d with 2 lanes and 250 MB of RAM.

- Backend server authentication, that takes 0.5 seconds on a 2 GHz CPU using 4 cores -- Argon2d with 8 lanes and 4 GB of RAM.

Argon2i is optimized for more realistic settings, where the adversary possibly can access the same machine, use its CPU or mount cold-boot attacks. We use three passes to get rid entirely of the password in the memory. We suggest the following settings:

- Key derivation for hard-drive encryption, that takes 3 seconds on a 2 GHz CPU using 2 cores - Argon2i with 4 lanes and 6 GB of RAM

- Frontend server authentication, that takes 0.5 seconds on a 2 GHz CPU using 2 cores - Argon2i with 4 lanes and 1 GB of RAM.

We recommend the following procedure to select the type and the parameters for practical use of Argon2.

1. Select the type y. If you do not know the difference between them or you consider side-channel attacks as viable threat, choose Argon2i.

2. Figure out the maximum number h of threads that can be initiated by each call to Argon2.

3. Figure out the maximum amount m of memory that each call can afford.

4. Figure out the maximum amount x of time (in seconds) that each call can afford.

5. Select the salt length. 128 bits is sufficient for all applications, but can be reduced to 64 bits in the case of space constraints.

6. Select the tag length. 128 bits is sufficient for most applications, including key derivation. If longer keys are needed, select longer tags.

7. If side-channel attacks is a viable threat, enable the memory wiping option in the library call.

8. Run the scheme of type y, memory m and h lanes and threads, using different number of passes t. Figure out the maximum t such that the running time does not exceed x. If it exceeds x even for t = 1, reduce m accordingly.

9. Hash all the passwords with the just determined values m, h, and t.

5. Example Code

TBD -- is there a python implementation?

6. Test Vectors

This section contains test vectors for Argon2.

6.1. Argon2d Test Vectors

```
=======================================Argon2d
Memory: 16 KiB
Iterations: 3
Parallelism: 4 lanes
Tag length: 32 bytes
Password[32]: 01 01 01 01 01 01 01 01
              01 01 01 01 01 01 01 01
              01 01 01 01 01 01 01 01
          01 01 01 01 01 01 01 01
Salt[16]: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
Secret[8]: 03 03 03 03 03 03 03 03
Associated data[12]: 04 04 04 04 04 04 04 04 04 04 04 04
Pre-hashing digest: ec a9 db ff fa c9 87 5c
                    d2 dc 32 67 cb 82 7f 48
            79 af db 2f 6c b3 a5 29
            c5 87 7c 60 7d 72 92 02
            7c 23 15 47 fc 64 4f b8
            81 16 1f ee f6 e2 b3 d1
            63 49 1a 98 e8 a8 8c 8a
            40 15 b8 b5 dc 85 ec 1b

 After pass 0:
Block 0000 [  0]: 7ddae3a315a45d2d
Block 0000 [  1]: 50d8b9a49514a996
Block 0000 [  2]: d5fd2f56c5085520
Block 0000 [  3]: 81fa720dcf94e004
...
Block 0031 [124]: 40b2d44e241f7a2a
Block 0031 [125]: 9b9658c82ba08f84
Block 0031 [126]: 917242b2a7a533f2
Block 0031 [127]: 4169db73ebcc9e9c

 After pass 1:
Block 0000 [  0]: a8daed017254d662
Block 0000 [  1]: 1564d0fc4f5d07f4
Block 0000 [  2]: 6a18ece1fd7d79ff
Block 0000 [  3]: d04eb389a8ac7324
...
```

```
   Block 0031 [124]: c859e8ba37e79999
   Block 0031 [125]: 0bb980cfe6552a4d
   Block 0031 [126]: 300cea2895f4459e
   Block 0031 [127]: 37af5d23a18f9d58

    After pass 2:
   Block 0000 [  0]: e86fc8e713dbf6d3
   Block 0000 [  1]: b30f1bdf8b4219d6
   Block 0000 [  2]: a84aec198d1eaff0
   Block 0000 [  3]: 1be35c5c8bfc52e0
   ...
   Block 0031 [124]: 9ffab191789d7380
   Block 0031 [125]: 4237012fc73e8d3e
   Block 0031 [126]: fbea11160fe7b50e
   Block 0031 [127]: 692210628c981931

   Tag: 57 b0 61 3b fd d4 13 1a
        0c 34 88 34 c6 72 9c 2c
        72 29 92 1e 6b ba 37 66
        5d 97 8c 4f e7 17 5e d2
```

6.2. Argon2i Test Vectors

```
   =======================================Argon2i
   Memory: 16 KiB
   Iterations: 3
   Parallelism: 4 lanes
   Tag length: 32 bytes
   Password[32]: 01 01 01 01 01 01 01 01
                 01 01 01 01 01 01 01 01
                 01 01 01 01 01 01 01 01
             01 01 01 01 01 01 01 01
   Salt[16]: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
   Secret[8]: 03 03 03 03 03 03 03 03
   Associated data[12]: 04 04 04 04 04 04 04 04 04 04 04 04
   Pre-hashing digest: c0 4e 5c 19 98 fc b1 12
                       09 3e 36 a0 76 3e 2f 95
               57 f2 cf 53 6f b8 89 c9
               9c c6 d8 cd b3 49 cd 0c
               9d 48 db cc 94 57 59 8c
               6c 2d a1 e1 d1 8b 3b aa
               7a 37 43 cb d1 7a d8 5c
               61 df dc 7e 7a 8e 64 2f

    After pass 0:
   Block 0000 [  0]: 34e7ba2a71020326
   Block 0000 [  1]: 3a4e252bf033a4cb
   Block 0000 [  2]: 3fb8e27bb8ab6a2b
```

```
Block 0000 [  3]: 65bb946635366867
...
Block 0031 [124]: 433d8954deddd5d6
Block 0031 [125]: c76ead72f0c08a23
Block 0031 [126]: b7c6ce1154c1fdd1
Block 0031 [127]: 0e766420b2ee181c

 After pass 1:
Block 0000 [  0]: 614a404c54646531
Block 0000 [  1]: 79f220080bfac514
Block 0000 [  2]: e9da047d0e4406b4
Block 0000 [  3]: 0995bc6d95590353
...
Block 0031 [124]: 9b89e743afa7b916
Block 0031 [125]: 9b3f7ca7cfff2db9
Block 0031 [126]: 0065ff067978eab8
Block 0031 [127]: 0a78fa2cea2b8bb2

 After pass 2:
Block 0000 [  0]: 3fea10517d1a7476
Block 0000 [  1]: e44c8bece4b3ecb2
Block 0000 [  2]: e348b27d988671cb
Block 0000 [  3]: 5f7f7cd33ef59e4d
...
Block 0031 [124]: f60cb937689b55f8
Block 0031 [125]: 418c55d7f343df3f
Block 0031 [126]: 26899dd11adc7474
Block 0031 [127]: dd3afa472ff1d124
Tag: 91 3b a4 37 68 5b 61 3c
     f1 2b 94 46 79 53 40 37
     ac 46 cf a8 8a 02 f6 c7
     ba 28 0e 08 89 40 19 f2
```

7. Acknowledgements

TBA

8. IANA Considerations

None.

9. Security Considerations

TBA

10. References

10.1. Normative References

[I-D.saarinen-blake2]
Saarinen, M. and J. Aumasson, "The BLAKE2 Cryptographic Hash and MAC", draft-saarinen-blake2-06 (work in progress), August 2015.

10.2. Informative References

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[ARGON2] Biryukov, A., Dinu, D., and D. Khovratovich, "Argon2: the memory-hard function for password hashing and other applications", WWW https://password-hashing.net/argon2-specs.pdf, October 2015.

Authors' Addresses

Alex Biryukov
University of Luxembourg

Daniel Dinu
University of Luxembourg

Dmitry Khovratovich
University of Luxembourg

Simon Josefsson
SJD AB

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