ACurless Key Derivation Function

Adrian Curless

March 2020

1 Overview

ACurless KDF has two distinct phases, each of which will be described in detail in this document. In the section for each phase, the computation is described, as well as a description of any magic constants in use.

The two phases are combined with an variable length input to produce a 32 bit key, as described below.

$$\text{KEY} = \text{PHASE2 (PHASE1 (input, iv))}$$

2 Phase One

Phase one is based upon the well known Blowfish cipher. Phase one, like Blowfish operates using a block size of 64 bits, which means it is well suited to implementation on modern CPUs. See Section 2.2 for a description of the modifications to Blowfish.

Phase One produces O, which is half the length of input, given input and an initialization vector iv, which always takes the value 0xC0FFEE.

$$O = \text{PHASE1(input, iv)}$$

The KDF utilizes Blowfish in a slightly unconventional way that is described in this section. See Section 2.1 for details regarding the sboxes and P-Array.

Perform the following steps to complete Phase One:

1. The KDF input (a character array) must be padded so the length can be evenly divided into blocks of 64 bits. Padding is simple, just fill the needed space with 0-bytes (0x00).
2. Divide the input data into 64 bit blocks.
3. Initialize the key as described in Section 2.3.
4. For each block of input:
   - Perform a single ACurless KDF key transformation as described in Section 2.4.
   - Initialize a Blowfish key using the output of the key transformation obtained above.
   - Blowfish encrypt the current block of input.
   - Append the upper 32 bits of the output to O.
   - Use the lower 32 bits of the output as the IV for the next block.

To be clear, at the end of phase one will output half as many blocks as it is given. If you provided 10 blocks of input, there will be 5 blocks of output (because only half of each block is uses as output).

2.1 P-Array and Substitution Boxes

ACurless KDF uses the same sboxes as Blowfish, we can be easily obtained online. Since Blowfish has a large sbox table, the table is omitted here for brevity. In addition to the sboxes, Blowfish makes use of a so called 'P-Array', the contents of which are also available online.

Both the sboxes and so called P-Array are composed of hexadecimal digits derived from Pi as stated in the Blowfish specification.

2.2 Modifications to Blowfish

The modifications made to the Blowfish cipher are enumerated here.

2.2.1 Encryption

In the encryption function, the function f is applied on both the upper and lower 32 bits of the block of input data. When the function f is applied to the upper 32 bits, a bitwise and should be computed from the output of f and the bitwise complement of the hexadecimal constant 0xFC. When the function f is applied to the lower 32 bits, a bitwise and should be computed of the output of f and the bitwise and of the same constant.

In the same function, the upper 32 bits of the input block is xored with the second element of the 'P-Array', instead of the 17th as done in standard blowfish. Similarly, the lower 32 bits are xored with the first element of the 'P-Array', as opposed to the 18th.

2.2.2 Function f

When looking up values in the odd numbered sboxes (1 and 3), a left circular shift of 2 is applied to the results prior to their usage. Similarly, a right circular shift of 3 is applied to the values looked up from the even numbered sboxes (2 and 4).

2.2.3 Key Size

In this modified version of Blowfish the key size is extended to 64 bytes.

2.3 Key Initialization

The initial key for Blowfish is a 64 byte sequence of random data. The key is shown in Listing 1.

Listing 1: Initial key used by Blowfish in ACurless KDF.

```
static uint8_t bf_initial_key[64] = {
    0x83, 0x25, 0x31, 0xdc, 0x3a, 0x51, 0xd5, 0x56, 0xc9, 0x09, 0x37,
    0x30, 0x97, 0xce, 0x50, 0x05, 0x5b, 0xc4, 0x1a, 0x9f, 0x74, 0x17,
    0x82, 0x35, 0x64, 0x00, 0xfe, 0xac, 0x0d, 0x88, 0x18, 0xe2, 0x42,
    0x88, 0x3c, 0x0f, 0x1a, 0xe3, 0x6a, 0x96, 0xfe, 0x73, 0xe3, 0x67,
    0x21, 0x36, 0xf5, 0x94, 0x73, 0x0d, 0x94, 0x73, 0x9d, 0x55, 0xca,
    0x7f, 0xde, 0xad, 0xbe, 0xef, 0xc0, 0xfe, 0xff, 0x01
};
```

2.4 ACurless KDF Key Transformation

- Divide the key into 32 bit chunks
- Replace each chunk of the key to the Exclusive Or of the chunk and the IV.

3 Phase Two

Phase Two accepts the output of Phase One and produces a 32 bit key (regardless of the input length).

$$\text{KEY} = \text{PHASE2}(O)$$

Phase Two is a sponge function, designed to take the variable length O produced in Phase One and make it fixed length (32 bits). In general, as is the case in ACurless KDF, a sponge function is composed of three components.

s: State containing 64 bits in ACurless KDF.

f: A function that transforms s using a pseudorandom permutation. See Section 3.2.

The state s is divided into two sections, the bitrate r and the capacity c. For performance and simplicity, r and c are both 32 bits, each one half of s. The lower 32 bits of s are assigned to r.

3.1 Sponge Operation

- s is initialized to 0.
- The input is divided into 32 bit blocks (The output of phase one is conveniently a multiple of 32 bits).
- For each block B of input,
  - $r = r \oplus B$
  - $s = f(s)$

  This is the absorption process of the 'sponge'.
- Now in the final step the output is 'squeezed' out:
  - $s = f(s)$
  - r is yielded as the output.

3.2 Pseudorandom function f

The pseudorandom function f is implemented as a linear congruential generator (LCG). The function f takes a 64 bit input and produces a 64 bit output (i.e. the function is modulo 2^{64}).

The LCG can be defined in terms of the recurrence relation:

$$X_{n+1} = (aX_n + b) \bmod m$$

where $a = 2862933555777941757$, $b = 3037000493$, $m = 2^{64}$.

4 Test Vectors

The following are official test vectors for ACurless KDF.

Input String	Key
Secret String	0x56ad4102
GNU+Linux	0x306d9005
Systemd is the worst	0x2146b411
Install Gentoo	0x6d707270