

---

# ciphart

---

## key derivation with easier measurable security

caveman  
January 9, 2021

---

*argon2*<sup>1</sup> is mostly nice, but trying to interpret its contribution to enhancing the protection against password brute-forcing attacks remains more difficult than it should be. this is a problem that is also shared with every other key derivation function that i've known so far.

when one uses *argon2*, his derived key will surely have superior protection against password brute-forcing attacks, but by how much? to answer this, one would need to survey the industry that manufactures application-specific integrated circuits to obtain a map between *time* and *money*. the centre of my thesis is that this part is not nice, because i found that life can be easier.

henceforth, i propose *ciphart* — a memory-hard key derivation function with a security contribution that is measured in a unit that i propose: *relative entropy bits*. this unit is measured objectively and is guaranteed to be true irrespective of whatever alien technology the adversary might have.

*libciphart*<sup>2</sup> — a library that implements *ciphart* very closely to this paper, without much fluff. this should make integrating *ciphart* into other systems more convenient. *ciphart*<sup>3</sup> — a tool for encrypting and decrypting files that makes use of *libciphart*. this tool is meant to be used by humans or scripts.

## 1 ciphart

### 1.1 parameters

- enc encryption function. this also defines  $C$ . e.g. if enc is *xchacha20*, then  $C = 64$  bytes to reflect its block size.
- $p$  password.
- $s$  salt.
- $L$  number of memory lanes for concurrency.
- $M$  total memory in bytes, increased to nearest multiple of  $2CL$ .
- $T$  number of  $C$ -byte tasks per lane segment.
- $R$  number of rounds per task.
- $B$  added security in *relative entropy bits*.
- $K$  output key size in bytes.

### 1.2 internal variables

- $\hat{M} \leftarrow M - (M \bmod 2CL) + 2CL$ . this is to ensure that it is in multiples of  $2CL$ .
- $G \leftarrow \hat{M}/C/T/L$ ; total number of segments per lane.
- $\hat{B} \leftarrow \begin{cases} B & \text{if } B \geq \log_2(GLTR) \\ \log_2(GLTR) & \text{otherwise} \end{cases}$   
this is to ensure that  $\hat{B}$  is large enough to have at least one pass over the  $\hat{M}$  bytes memory.
- $m_i$   $C$  bytes memory for  $i^{th}$  task in  $\hat{M}$ -bytes pad.
- $n_l \leftarrow lG$ ; nonce variable for  $l^{th}$  lane with at least 64 bits.
- $f \leftarrow \text{false}$ ; a flag indicating whether the  $\hat{M}$ -byte pad is filled.

### 1.3 output

- $k$   $K$ -bytes key with  $\geq B$  *relative entropy bits*.

---

**algorithm 1:** ciphart version 6

---

```
1 while true do
2   for  $g = 0, 1, \dots, G - 1$  do
3     for  $l = 0, 1, \dots, L - 1$  do
4       for  $t = 0, 1, \dots, T - 1$  do
5         for  $r = 0, 1, \dots, R - 1$  do
6            $i \leftarrow gT + t$ ;
7           if  $t = 0$  then
8              $j \leftarrow i + T - 1$ ;
9           else if  $t = T - 1$  then
10             $j \leftarrow i + 1 - T$ ;
11          else
12             $j \leftarrow i + 1$ ;
13           $m_j \leftarrow \text{enc}(m_i, n_l, k)$ ;
14           $n_l \leftarrow n_l + 1$ ;
15           $k \leftarrow f(m_j, p, l, s, t)$ ;
16        if  $f = \text{true}$  and  $\log_2(n_l L) \geq B$  then
17          go to line 19;
18       $f \leftarrow \text{true}$ ;
19 while true do
20   for  $l = 1, 2, \dots, L$  do
21     if  $\text{len}(k) \geq K$  then return  $k[0 : K]$ ;
22      $n \leftarrow n + 1$ ;
23      $k \leftarrow k \parallel \text{enc}(m_{l,S,T}[1], n, k)$ ;
```

---

<sup>1</sup><https://github.com/P-H-C/phc-winner-argon2>

<sup>2</sup><https://github.com/Al-Caveman/libciphart>

<sup>3</sup><https://github.com/Al-Caveman/ciphart>

- 2 parallelism
- 3 memory-hardness
- 4 security interpretation
- 5 comparison
- 6 summary