key derivation with easier measurable security

caveman

January 8, 2021

hi — i propose *ciphart*, a sequential memory-hard key derivation function that has a security gain that's measurable more objectively and more conveniently than anything in class known to date.

to nail this goal, *ciphart*'s security gain is measured in the unit of *relative entropy bits*. relative to what? relative to the encryption algorithm that's used later on. therefore, this *relative entropy bits* measure is guaranteed to be true when the encryption algorithm that's used with *ciphart* is also the same one that's used to encrypt the data afterwards.

my reference implementation is available here[1].

content

[1] `https://github.com/Al-Caveman/ciphart`

1 ciphart

parameters:

W	each task's size, at least 32 bytes.
M	total memory in multiples of $2W$.
R	number of rounds per task.
B	added security in *relative entropy bits.*
enc	encryption function.
k	initial key.

input:

T	$\leftarrow M/W$
P	$\leftarrow \max(2, \lceil 2^B/(TR) \rceil)$
$m_{l,s,t}$	memory for t^{th} task, in s^{th} segment, in l^{th} lane to work on.
n	$\leftarrow 0$, a variable with enough bytes to store nonces in. $n[0]$ means first 64 bits. $n[1]$ means second 64 bits.

output:

$\hat{k}$	better key, with B, or more, *relative entropy bits.*

Algorithm 1: ciphart version 6

```
1  while true do
2      for s = 1, 2, ..., S do
3          for l = 1, 2, ..., L do
4              for t = 1, 2, ..., T do
5                  for r = 1, 2, ..., R do
6                      if t = 1 then
7                          m_{l,s,t} ← enc(m_{l,s,T}, n, k);
8                      else
9                          m_{l,s,t} ← enc(m_{l,s,t−1}, n, k);
10                     n[1] ← n[1] + 1;
11                     k ← f(m_{l,s,t}[−64 :], p, l, s, t);
12         if log_2 (n[0] × SLTR + n[1]) ≥ B then
13             go to step 15;
14     n[0] ← n[0] + 1;
15 while true do
16     n[0] ← n[0] + 1;
17     for l = 1, 2, ..., L do
18         if len(k̂) ≥ K then
19             return k̂[0 : K]
20         k̂ ← k̂ || enc(m_{l,S,T}[1], n, k);
21         n[1] ← n[1] + 1;
```

2 parallelism

3 hardness with smaller memory

4 security interpretation

5 comparison

6 summary