key derivation with easier measurable security

caveman

January 1, 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:

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

input:

T	$\leftarrow W/M$
P	$\leftarrow \max(2, \lceil 2^B/(TR) \rceil)$
x	$\leftarrow 0$, a 16 bytes wide variable.
m_t	for any task $t \in \{1, 2, \dots, T\}$, m_t is M-bytes memory for t^{th} task to work on. $m_t[0:16]$ means first 16 bytes. $m_t[-16:]$ means last 16 bytes.
nonce	a variable with enough bytes to store nonces in.
hash	a function to compress W bytes into desired key length.

output:

$\hat{k}$	better key, with B, or more, *relative entropy bits*. specifically, with $\log_2(PTR) \geq B$ bits.

steps:

for $p = 1, 2, \dots, P$ **do**
  for $t = 1, 3, \dots, T-1$, in steps of 2 **do**
    $i \leftarrow t$
    $j \leftarrow t + 1$
    for $r = 1, 2, \dots, 2R$ **do**
      nonce $\leftarrow (p, t, r)$
      $m_i \leftarrow \text{enc}(m_j, \text{nonce}, k)$
      $\hat{i} \leftarrow i$
      $i \leftarrow j$
      $j \leftarrow \hat{i}$
    end for
    $x \leftarrow x \oplus m_i[-16:]$
    $x \leftarrow x \oplus m_j[-16:]$
  end for
  for $t = 1, 2, \dots, T$ **do**
    $m_t[0:16] \leftarrow m_t[0:16] \oplus x$
  end for
end for
return $\hat{k} \leftarrow \text{hash}(m_1, m_2, \dots, m_T)$

2 parallelism

3 sequential-memory hardness

4 security interpretation

5 comparison

6 summary