



script 算法的原理和应用

@EtherDream 2017-01-17

scrypt 是什么?

scrypt 是一种 密码学 Hash 函数

专用于 Hash 口令

密码学 Hash 函数

常见的用途：

- 数据校验，例如 `SHA256`
- 数据签名，例如 `HMAC_SHA256`
- 口令处理，例如 `PBKDF2_SHA256`

口令特点

为了方便记忆，口令通常有一定的规律

- 123456、888888
- QWERT、asdfg
- 电话、生日、姓名等组合

口令破解

知道 哈希值 H 和 算法 F , 即可跑字典

```
for each x in dict
    if  $F(x) == H$  then
        cracked!
```

对抗破解

跑字典 是通用的攻击方式，无法避免

唯一可做的，就是：

提高 Hash 函数 计算成本，降低破解速度

普通 Hash 函数

MD5、SHA256、开发者修改的变种 ...

计算时间：1 μ s

破解速度：1,000,000 hash/s （单线程）

增加耗时

反复 Hash 大量次数，密码学称之 **拉伸**：

```
func slow_hash(x)
    for i = 0 to 1000000
        x = hash(x)
    return x
```


口令 Hash 函数

例如 PBKDF2 算法, 可设定 拉伸次数 N :

```
func PBKDF2(F, ..., N)
    ...
    for i = 0 to N
        ...
        x = F(x, ...)
        ...
    end
    ...
```

PBKDF2 缺陷

PBKDF2 算法简单，很容易被 GPU 并行计算

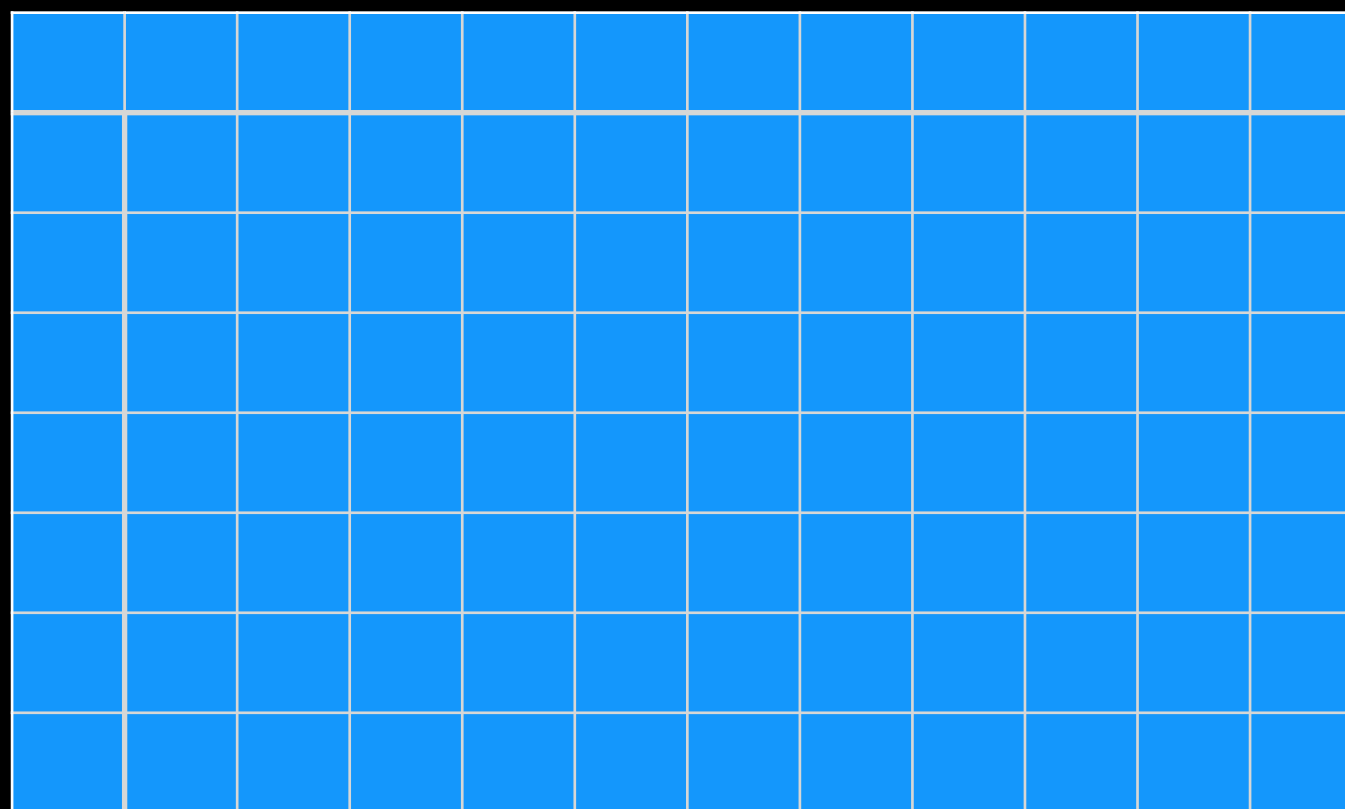
```
SHA256_Transform(uint32_t * state, const
{
    uint32_t W[64];
    uint32_t S[8];
    ...
    /* 3. Mix. */
    RNDr(S, W, 0, 0x428a2f98);
    RNDr(S, W, 1, 0x71374491);
    ...
    RNDr(S, W, 60, 0x90befffa);
    RNDr(S, W, 61, 0xa4506ceb);
    RNDr(S, W, 62, 0xbef9a3f7);
    RNDr(S, W, 63, 0xc67178f2);
```

并发破解

CPU:

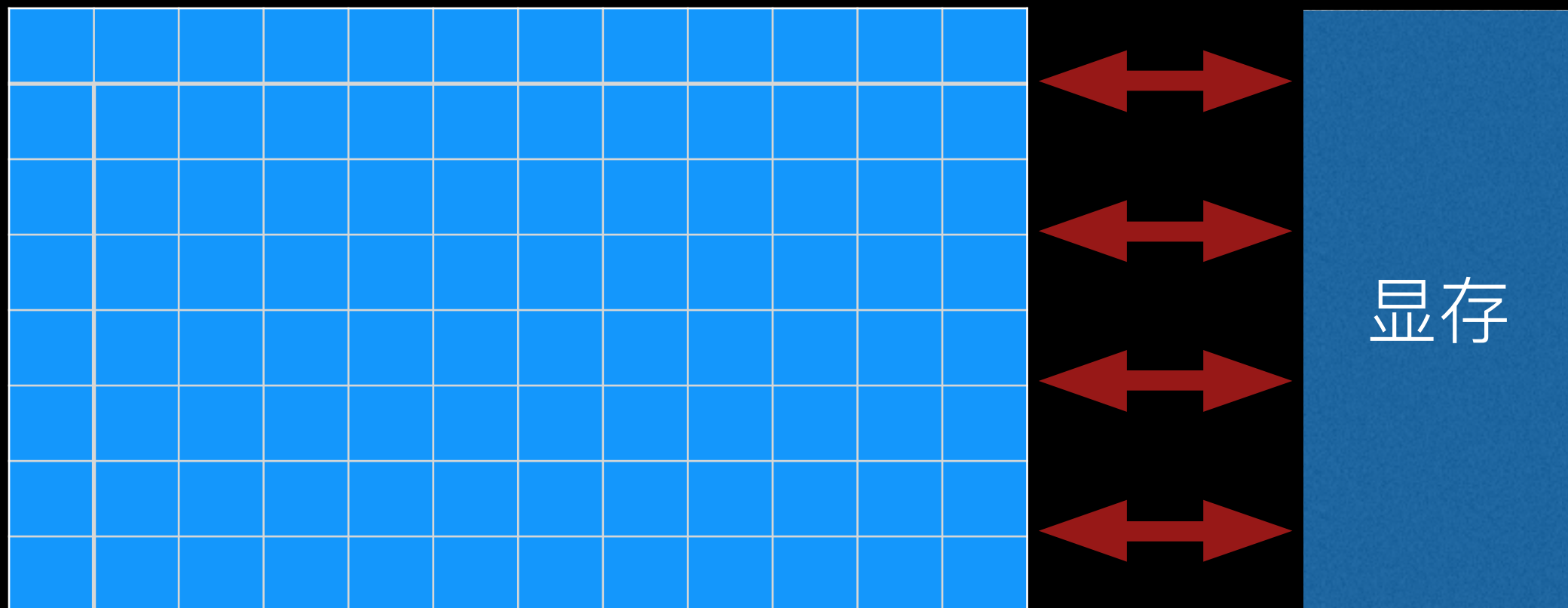


GPU:



对抗并发 - 方案 1

寻找 并发设备 的短板，例如 存储带宽



依赖存储

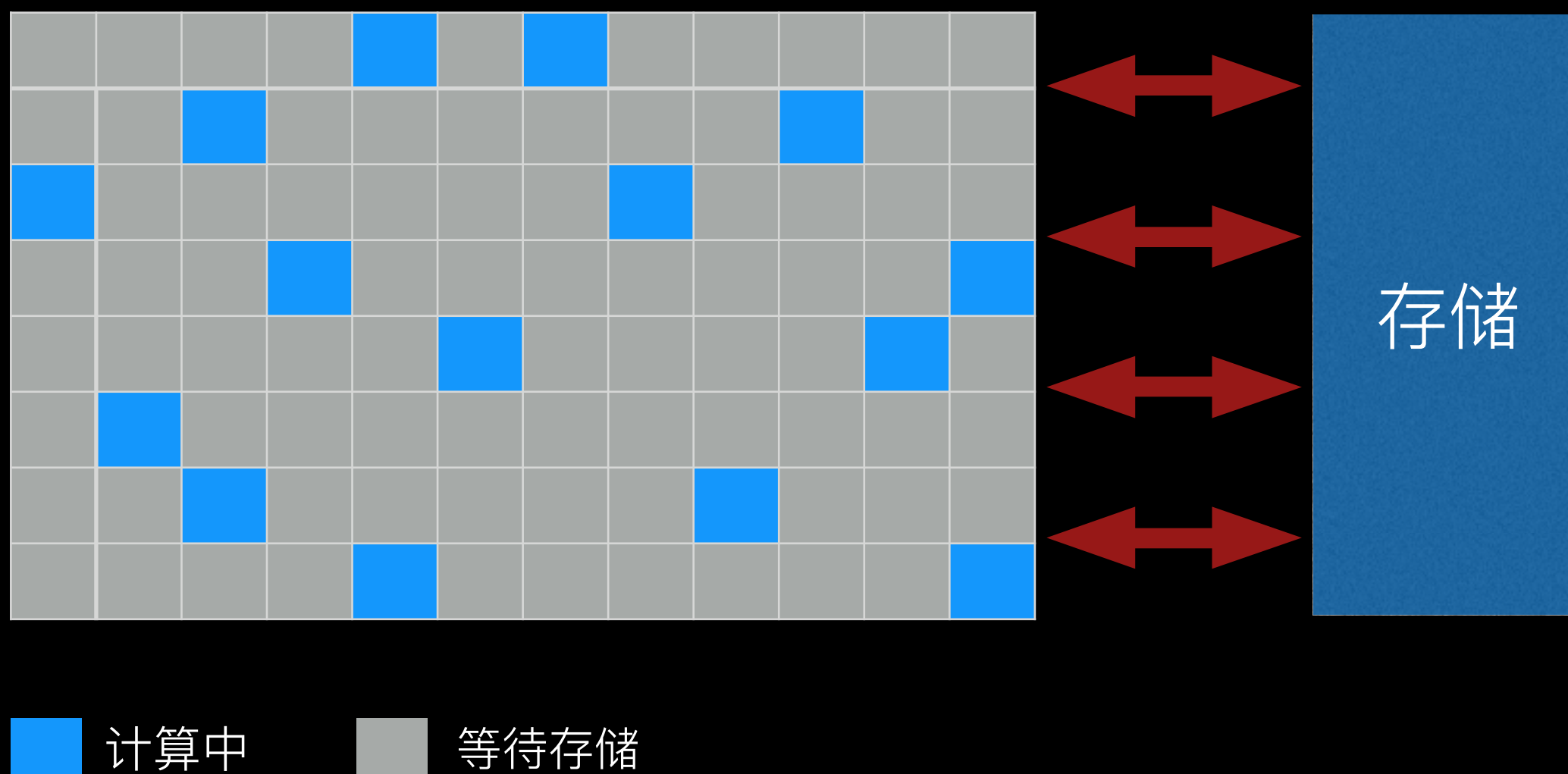
Memory-Hard Function

- 算法需要大量存储（GPU 线程多显存少）
- 频繁、随机读写（无法命中高速缓存）

最终瓶颈出现在 存储 上，计算力无法发挥

依赖存储

计算单元都在等存储，无法满负荷运行



依赖内存的原理

大量的 随机存储访问。例如：

```
int mem[N]

for i = 0 to 1000000

    x = int( hash(x) )    // 该值无法预测

    mem[x % N] += x

result = hash(mem)        // 结果依赖存储
```

对抗并发 - 方案 2

Hash 算法可并行计算，利用多核 CPU 增加工作量：

线程 1: $dk1 = \text{PBKDF}(\text{Pass}, \text{Salt} + \text{"1"})$

线程 2: $dk2 = \text{PBKDF}(\text{Pass}, \text{Salt} + \text{"2"})$

...

线程 N: $dkN = \text{PBKDF}(\text{Pass}, \text{Salt} + \text{N})$

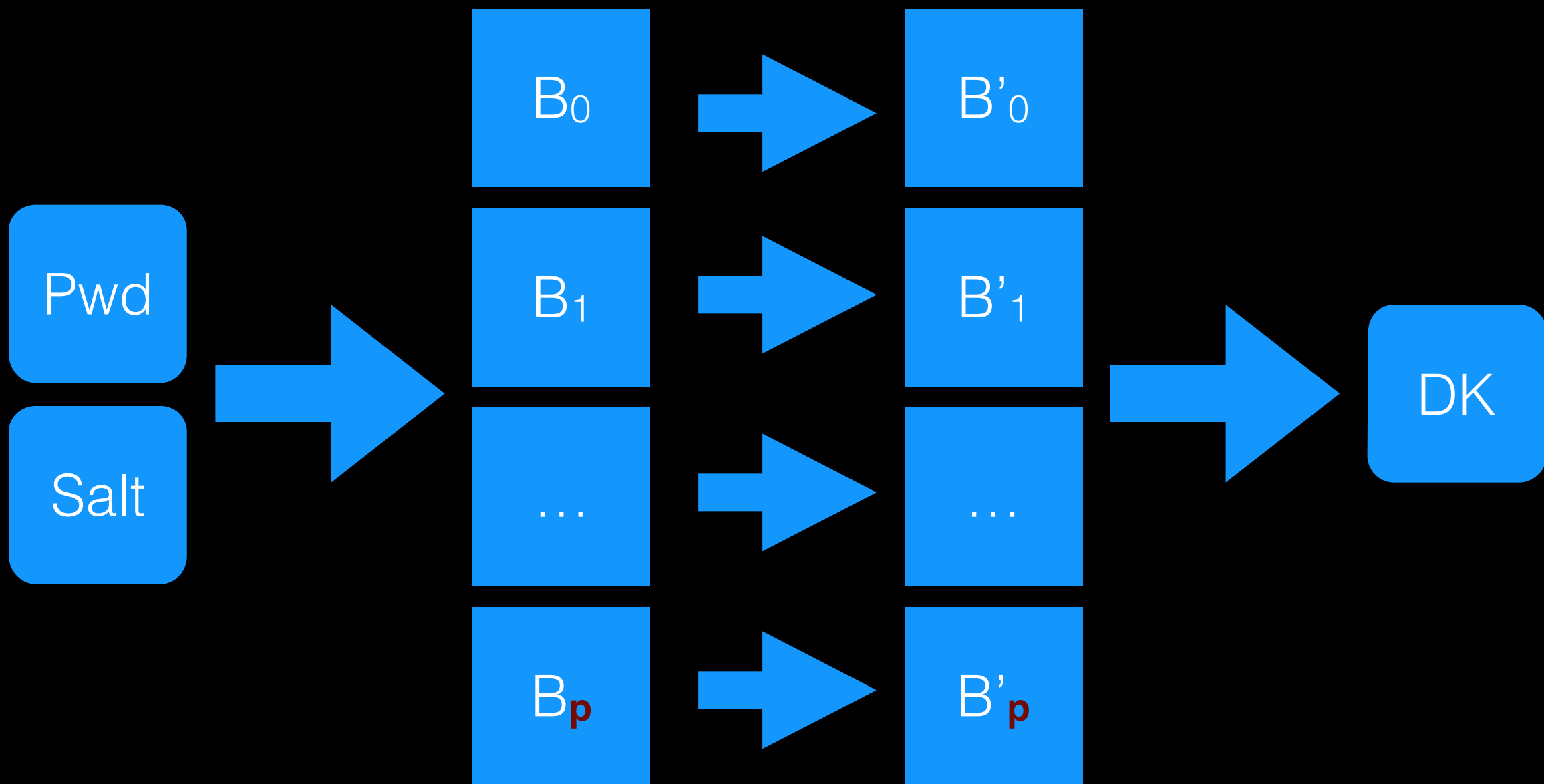
最终结果: $\text{Hash}(dk1 + dk2 + \dots + dkN)$

scrypt

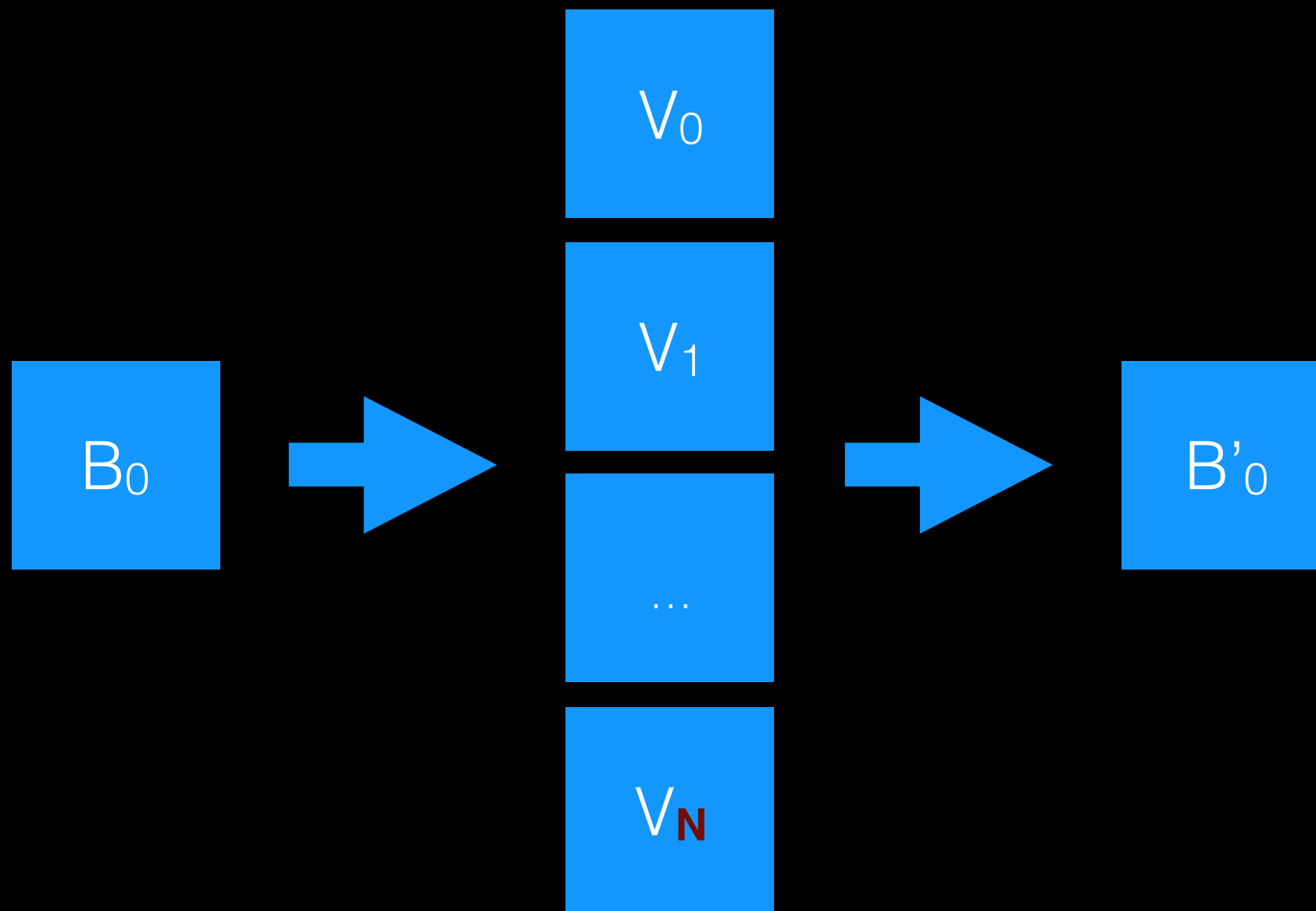
一种更先进的 Password Hash 函数。

- 支持 空间成本，对 GPU、ASIC 具有一定抗性
- 支持 并发维度，更好地利用 多线程 资源

并发维度 P



时空成本 N



细节

输入：

Pass、Salt、r（块长度）、N（成本）、P（并行）

第一步：

```
B = PBKDF2_SHA256(Pass, Salt, 1, r * 128 * P)
```

第二步：

```
for i = 0 to P - 1
```

```
    B'[i] = SMix( B[i], N, r )
```

输出：

```
dk = PBKDF2_SHA256(Pass, B', 1, dkLen)
```

scrypt 历史

Colin Percival 为备份服务 Tarsnap 设计。

客户端：

`dk` = `scrypt`(用户输入的口令)

`output` = `encrypt`(备份的文件, `dk`)

密文备份到云端

script 流行

运用于「密码学货币」工作量证明

- Tenebrix
- Litecoin (莱特币)
- Dogecoin (狗狗币)

更先进的 KDF

Argon2

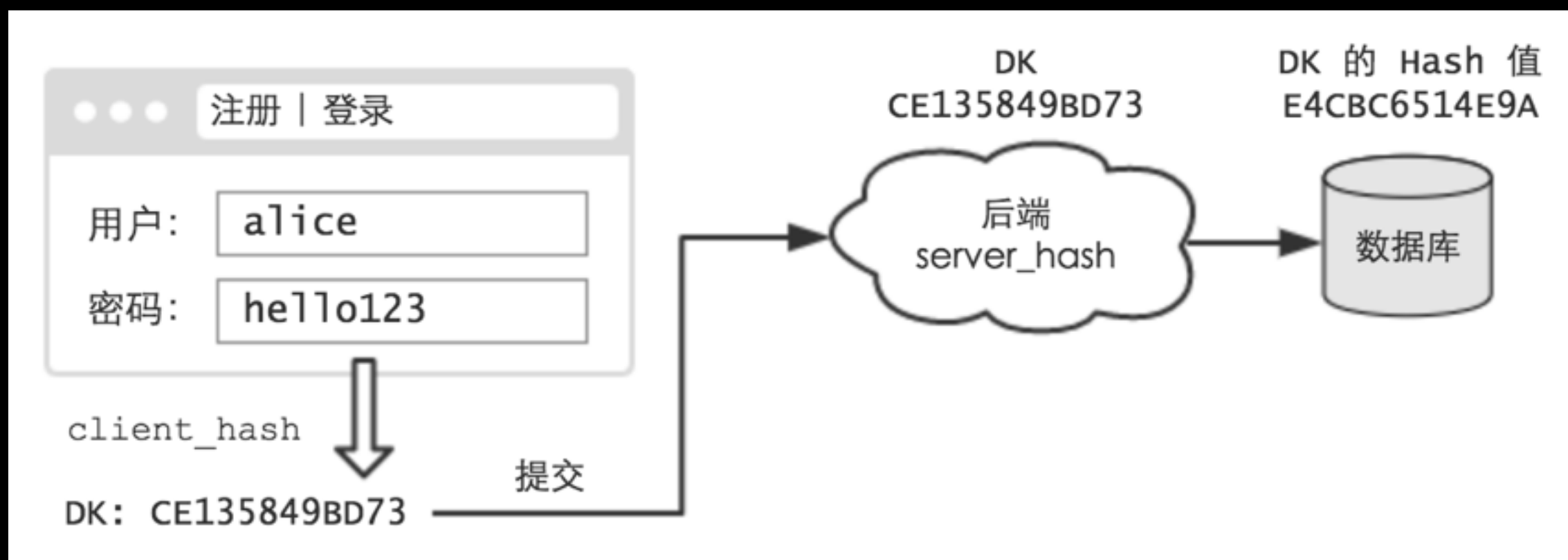
2015 年 Password Hashing Competition 获胜者

- 对内存依赖更充分
- 独立的时间、空间成本参数

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

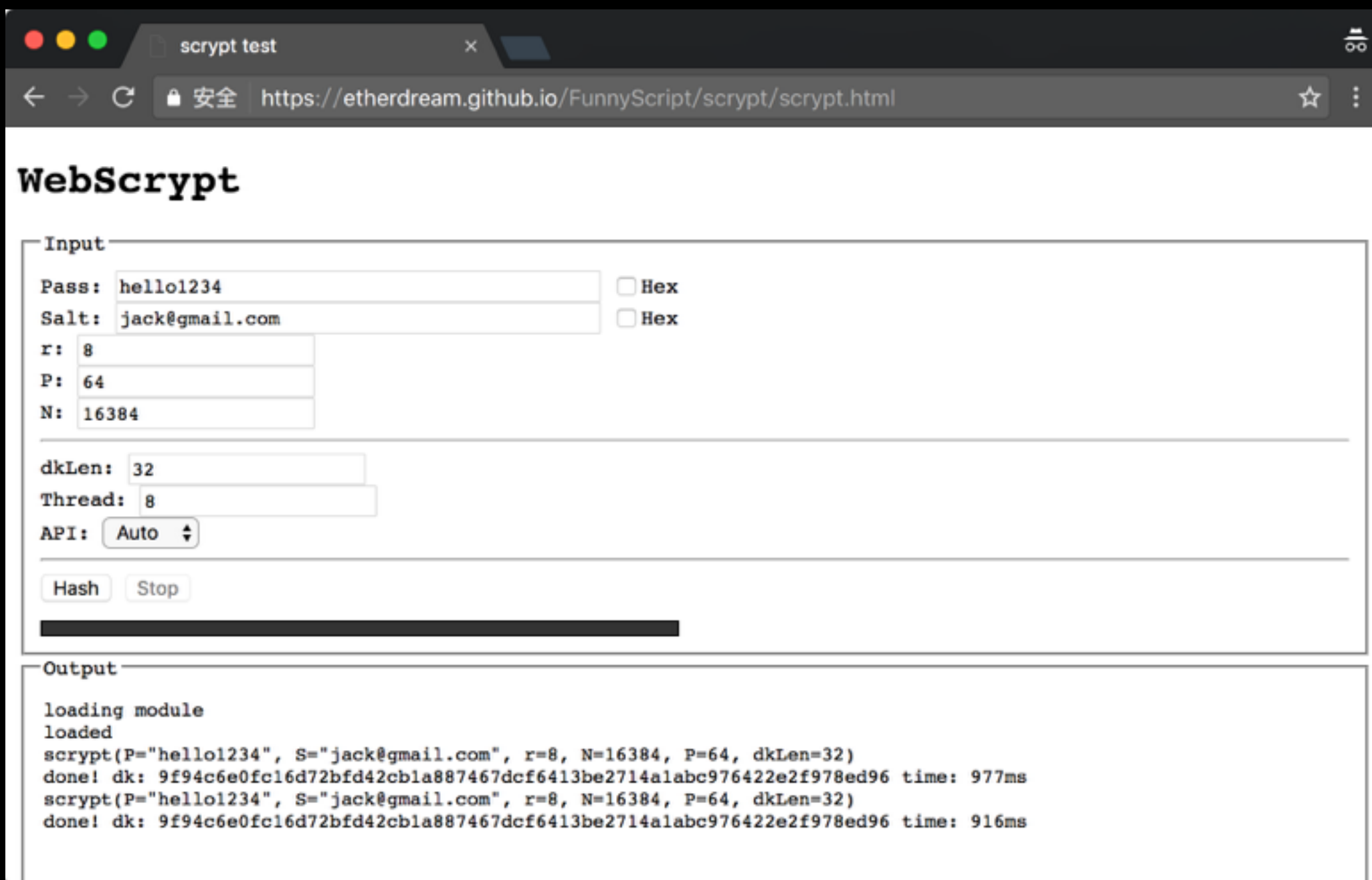
前端计算 KDF

KDF 很耗资源，可交给前端计算



算法移植

移植到浏览器的 scrypt 算法, 80% 的本地性能



script test

安全 https://etherdream.github.io/FunnyScript/scrypt/scrypt.html

WebScrypt

Input

Pass: ☐ Hex

Salt: ☐ Hex

r:

P:

N:

dkLen:

Thread:

API:

Output

```
loading module
loaded
scrypt(P="hello1234", S="jack@gmail.com", r=8, N=16384, P=64, dkLen=32)
done! dk: 9f94c6e0fc16d72bfd42cb1a887467dcf6413be2714alabc976422e2f978ed96 time: 977ms
scrypt(P="hello1234", S="jack@gmail.com", r=8, N=16384, P=64, dkLen=32)
done! dk: 9f94c6e0fc16d72bfd42cb1a887467dcf6413be2714alabc976422e2f978ed96 time: 916ms
```

WebScript

移植过程中的优化：

- 新浏览器 asm.js (emscripten)
- 老浏览器 Flash (FlasCC)
- 代码展开，尽可能用 JS 栈 取代 C 栈

<https://github.com/EtherDream/WebScript>

感谢观看