PHC: the candidates

JP Aumasson

KUDELSKI SECURITY

@veorq / http://aumasson.jp

academic background

principal cryptographer at Kudelski Security, .ch

applied crypto research and outreach

- BLAKE, BLAKE2, SipHash, NORX
- Crypto Coding Standard
- Password Hashing Competition
- Open Crypto Audit Project board member

no introduction

(bottom line: passwords' protection s****)

BLAME GAME

I didn't say it was your fault. I said I was going to blame you.

users using "weak passwords"?

ITsec people using "weak defenses"?

developers using "weak hashes"?

cryptographers, who never bothered?

agenda

1. the Password Hashing Competition (PHC)
2. the 24-2 PHC candidates
3. next steps, and how to contribute

WARNING

this is **NOT** about bikeshed topics as:

password policies

password managers

password-strength meters

will-technology-X-replace-passwords?

1. the Password Hashing Competition

another crypto competition
(cf. AES, eSTREAM, SHA-3, CAESAR)

try to survive and break the others

Tony Arcieri (@bascule, Square)

Jean-Philippe Aumasson (@veorq, Kudelski Security)

Dmitry Chestnykh (@dchest, Coding Robots)

Jeremi Gosney (@jmgosney, Stricture Consulting Group)

Russell Graves (@bitweasil, Cryptohaze)

Matthew Green (@matthew_d_green, Johns Hopkins University)

Peter Gutmann (University of Auckland)

Pascal Junod (@cryptopathe, HEIG-VD)

Poul-Henning Kamp (FreeBSD)

Stefan Lucks (Bauhaus-Universität Weimar)

Samuel Neves (@sevenps, University of Coimbra)

Colin Percival (@cperciva, Tarsnap)

Alexander Peslyak (@solardiz, Openwall)

Marsh Ray (@marshray, Microsoft)

Jens Steube (@hashcat, Hashcat project)

Steve Thomas (@Sc00bzT, TobTu)

Meltem Sonmez Turan (NIST)

Zooko Wilcox-O'Hearn (@zooko, Least Authority Enterprises)

Christian Winnerlein (@codesinchaos, LMU Munich)

Elias Yarrkov (@yarrkov)

Timeline

2013 Q1 call for submissions

2014 March 31 submission deadline

2014 Q3 selection of finalists

2015 Q2 selection of one or more winners https://password-hashing.net

https://password-hashing.net/wiki

discussions@password-hashing.net

#phc @freenode

2. the 24-2 PHC candidates

submissions requirements

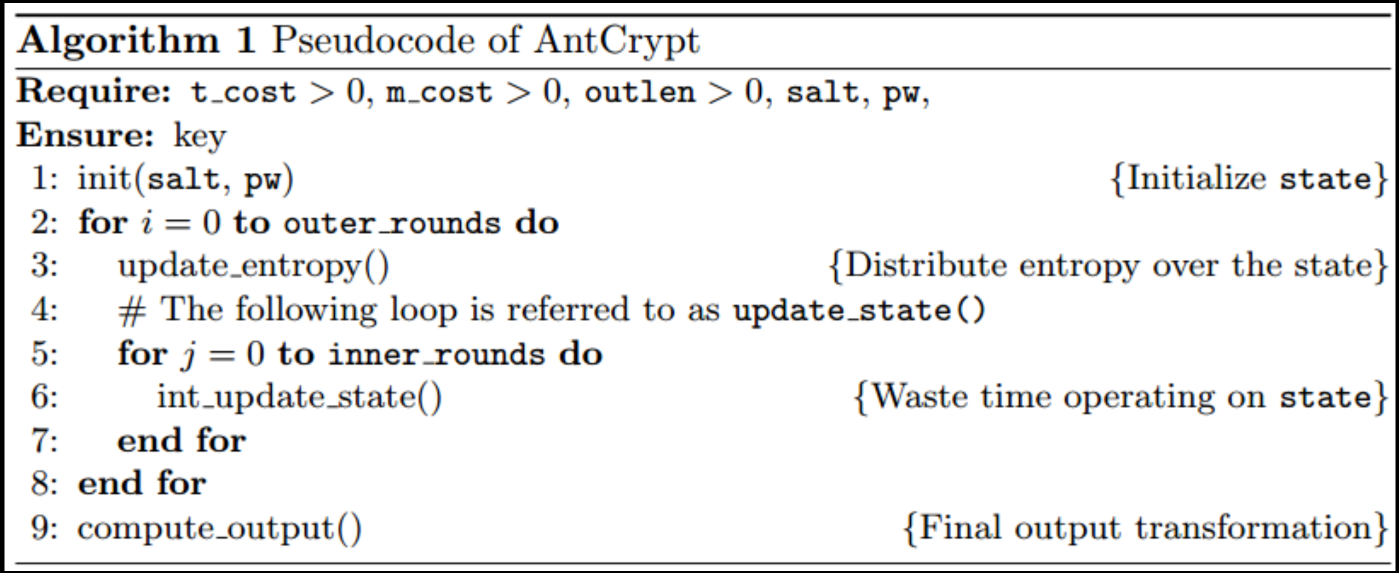
specs, reference code, test vectors

salt, time and memory parameters

IP statement: no patent, royalty-free

Antcrypt (Duermuth, Zimmerman)

- uses **SHA-512**
- **floating-point** arithmetic (pros and cons)
- separation crypto- and compute-hardness
- clear and well-motivated design

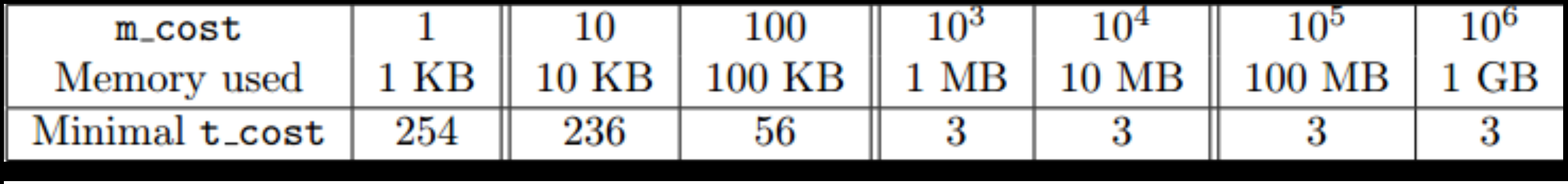


Algorithm 1 Pseudocode of AntCrypt

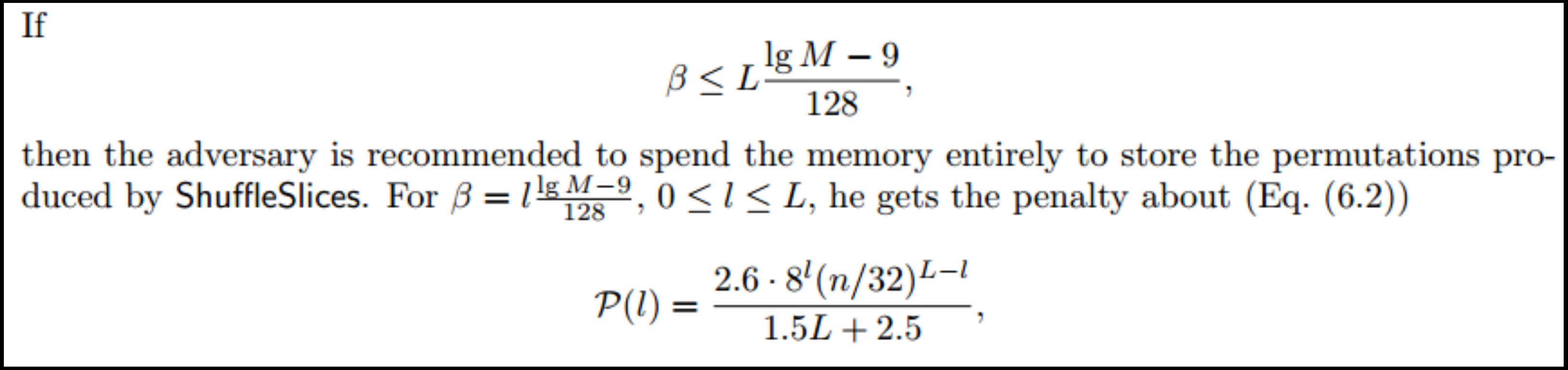
```
Require: t_cost > 0, m_cost > 0, outlen > 0, salt, pw,
Ensure: key
init(salt, pw)                                          {Initialize state}
for $i = 0$ to outer_rounds do
    update_entropy()                     {Distribute entropy over the state}
    # The following loop is referred to as update_state()
    for $j = 0$ to inner_rounds do
        int_update_state()                 {Waste time operating on state}
    end for
end for
compute_output()                              {Final output transformation}
```

Argon (Biryukov, Khovratovich)

- uses **AES-128** (thus NIs on defenders' CPUs)
- up to 32x parallelism, optional secret key
- supports **server relief** and **hash upgrade**
- thorough security analysis



m_cost	1	10	100	10^3	10^4	10^5	10^6
Memory used	1 KB	10 KB	100 KB	1 MB	10 MB	100 MB	1 GB
Minimal t_cost	254	236	56	3	3	3	3



If

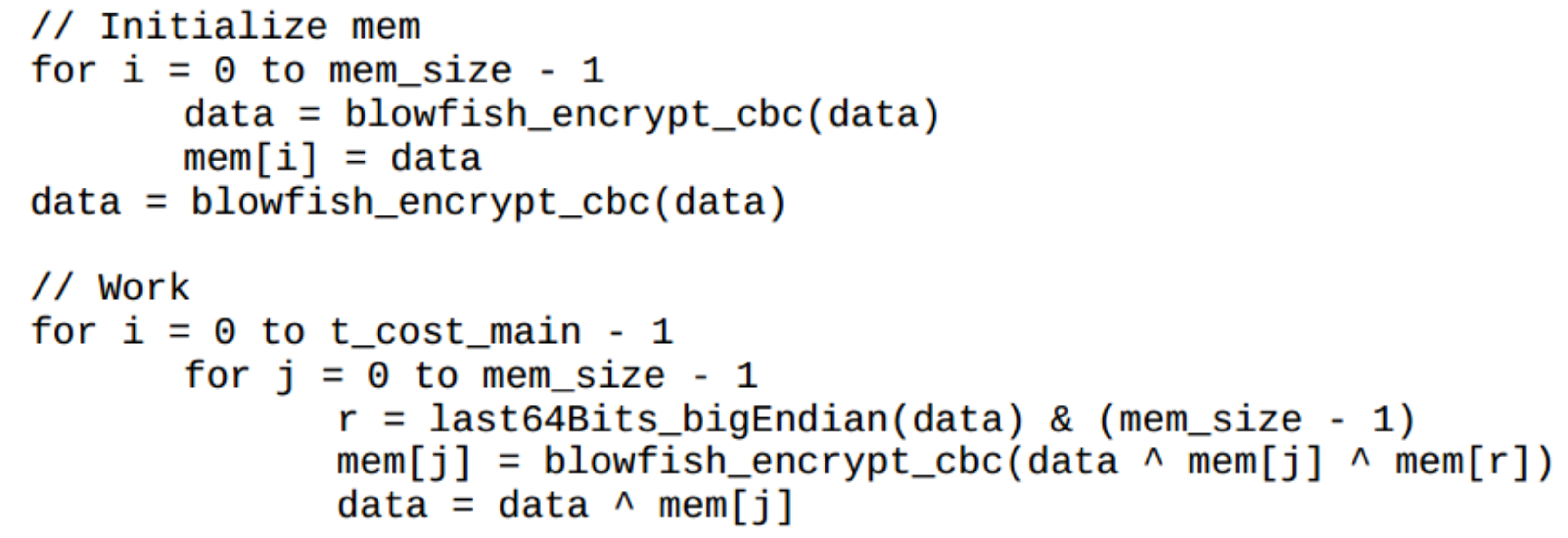
$$\beta \le L\frac{\lg M - 9}{128},$$

then the adversary is recommended to spend the memory entirely to store the permutations produced by ShuffleSlices. For $\beta = l\frac{\lg M-9}{128}$, $0 \le l \le L$, he gets the penalty about (Eq. (6.2))

$$\mathcal{P}(l) = \frac{2.6 \cdot 8^l (n/32)^{L-l}}{1.5L + 2.5},$$

battcrypt (Thomas)

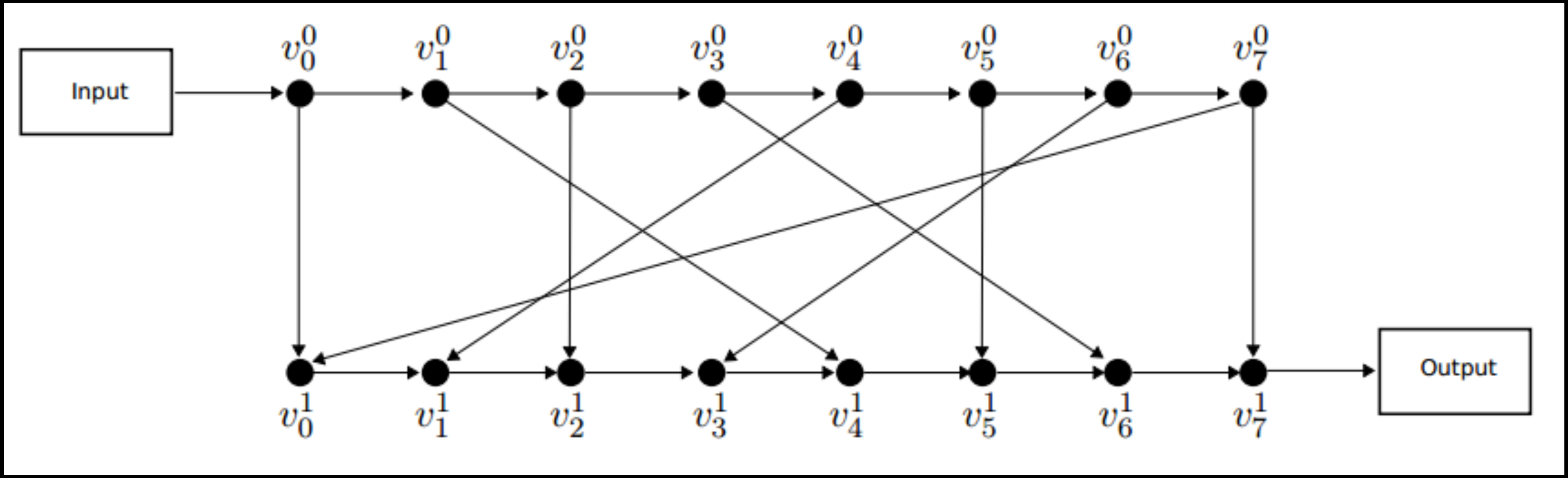
- **Blowfish** All The Things, and **SHA-512**
- suited for **PHP** (has a native Blowfish)
- supports **server relief** and **hash upgrade**
- elegant and minimalistic design



```
// Initialize mem
for i = 0 to mem_size - 1
       data = blowfish_encrypt_cbc(data)
       mem[i] = data
data = blowfish_encrypt_cbc(data)

// Work
for i = 0 to t_cost_main - 1
       for j = 0 to mem_size - 1
              r = last64Bits_bigEndian(data) & (mem_size - 1)
              mem[j] = blowfish_encrypt_cbc(data ^ mem[j] ^ mem[r])
              data = data ^ mem[j]
```

Catena (Forler, Lucks, Wenzel)

- uses **BLAKE2b** (thus SIMD on defenders' CPUs)
- **graph**-based structure, optional secret key
- supports **server relief** and **hash upgrade**
- thorough security analysis, and "proofs"

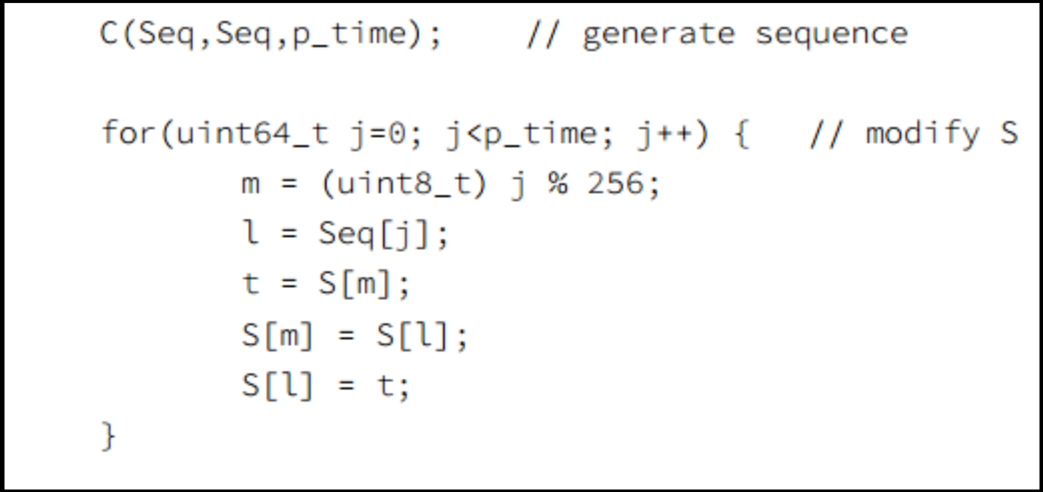


Catfish

WITHDRAWN

Centrifuge (Alvarez)

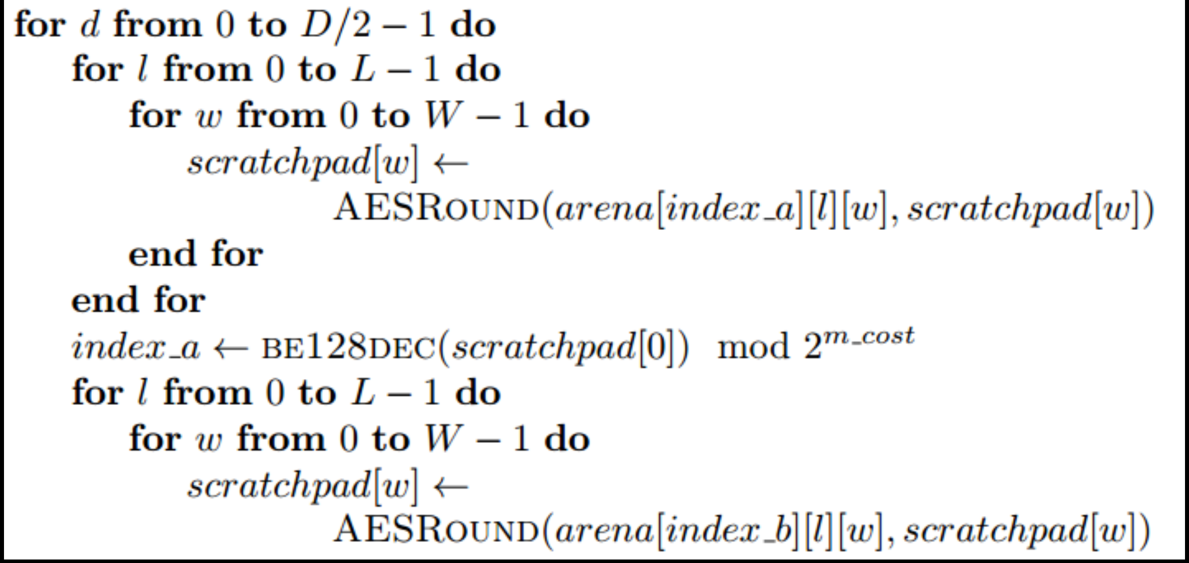
- uses **AES-256-CFB** and **SHA-512**
- benefits of **AES-NI** on defenders' CPUs
- password- and salt-dependent **"S-box"**
- RC4-like byte pseudorandom byte swap



```
C(Seq,Seq,p_time);    // generate sequence

for(uint64_t j=0; j<p_time; j++) {   // modify S
        m = (uint8_t) j % 256;
        l = Seq[j];
        t = S[m];
        S[m] = S[l];
        S[l] = t;
}
```

EARWORM (Franke)

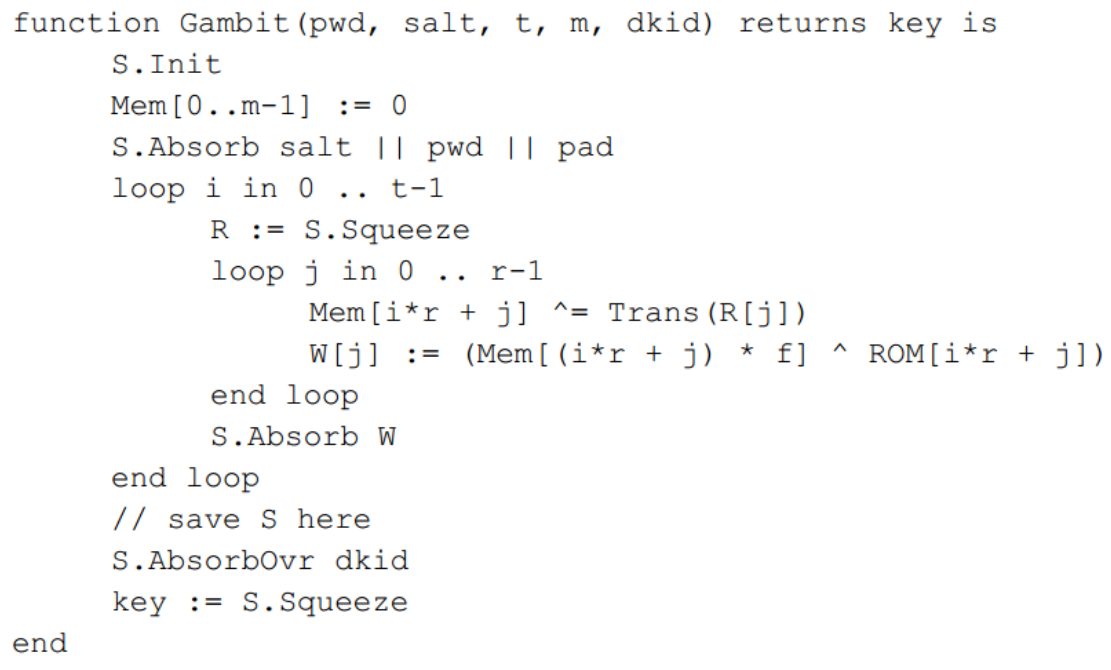
- uses **AES** round and **PBKDF2-HMAC-SHA-256**
- local **ROM** table (“arena”)
- not 2nd-preimage resistant (HMAC’s H(key)...)
- analysis wrt network timing attacks



```
for $d$ from $0$ to $D/2 - 1$ do
    for $l$ from $0$ to $L - 1$ do
        for $w$ from $0$ to $W - 1$ do
            $scratchpad[w] \leftarrow$
                AESRound($arena[index\_a][l][w], scratchpad[w]$)
        end for
    end for
    $index\_a \leftarrow$ BE128DEC($scratchpad[0]$)  mod $2^{m\_cost}$
    for $l$ from $0$ to $L - 1$ do
        for $w$ from $0$ to $W - 1$ do
            $scratchpad[w] \leftarrow$
                AESRound($arena[index\_b][l][w], scratchpad[w]$)
```

Gambit (Pintér)

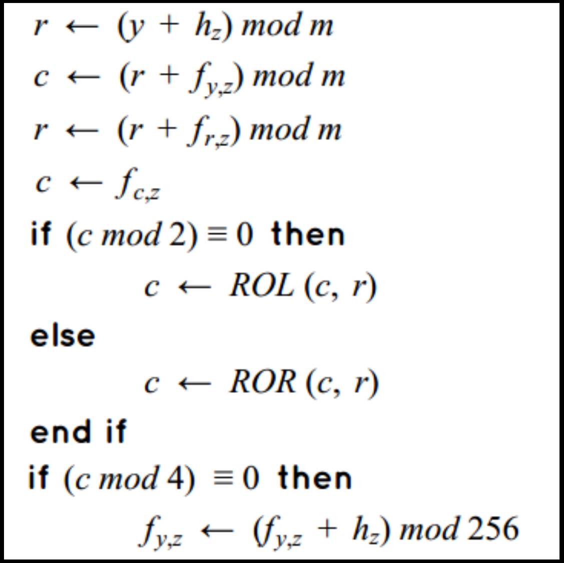
- uses **Keccak**[1600] (sponge function)
- optional local **ROM** table
- customizable word-to-word transform



```
function Gambit(pwd, salt, t, m, dkid) returns key is
      S.Init
      Mem[0..m-1] := 0
      S.Absorb salt || pwd || pad
      loop i in 0 .. t-1
            R := S.Squeeze
            loop j in 0 .. r-1
                  Mem[i*r + j] ^= Trans(R[j])
                  W[j] := (Mem[(i*r + j) * f] ^ ROM[i*r + j])
            end loop
            S.Absorb W
      end loop
      // save S here
      S.AbsorbOvr dkid
      key := S.Squeeze
end
```

Lanarea (Mubarak)

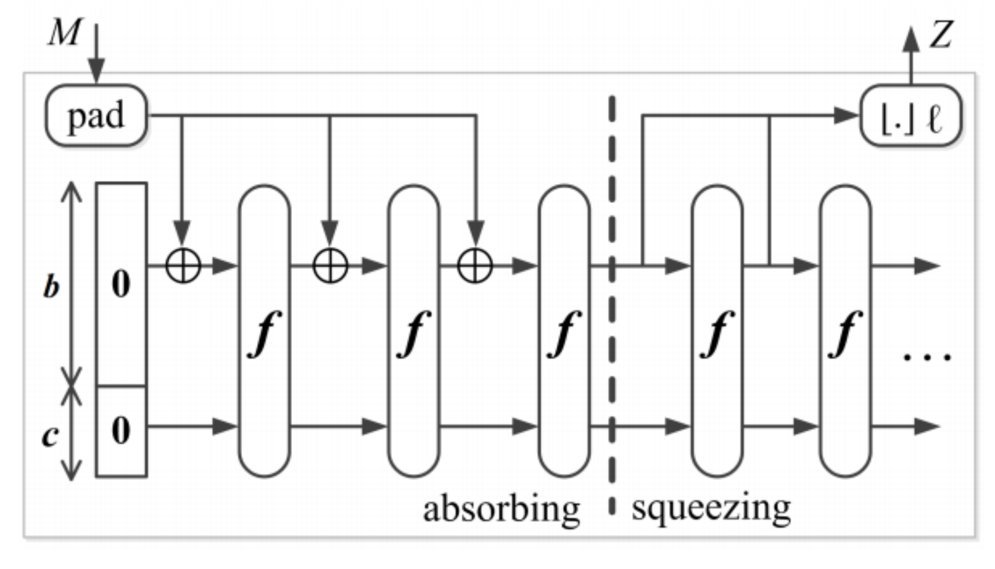
- uses **BLAKE2b**
- “heavily serial operations” (no //ism)
- “nonuniform section timings” (no pipelining)
- supports **hash upgrade**



```
$r \leftarrow (y + h_z)\ mod\ m$
$c \leftarrow (r + f_{y,z})\ mod\ m$
$r \leftarrow (r + f_{r,z})\ mod\ m$
$c \leftarrow f_{c,z}$
if $(c\ mod\ 2) \equiv 0$ then
        $c \leftarrow ROL\ (c,\ r)$
else
        $c \leftarrow ROR\ (c,\ r)$
end if
if $(c\ mod\ 4) \equiv 0$ then
        $f_{y,z} \leftarrow (f_{y,z} + h_z)\ mod\ 256$
```

Lyra2 (Simplicio Jr, Almeida, Andrade, dos Santos, Barreto)

- uses **BLAKE2b** (permut.) in a duplex sponge
- 2-dimensional memory parameter
- “basil” personalization string
- thorough security analysis

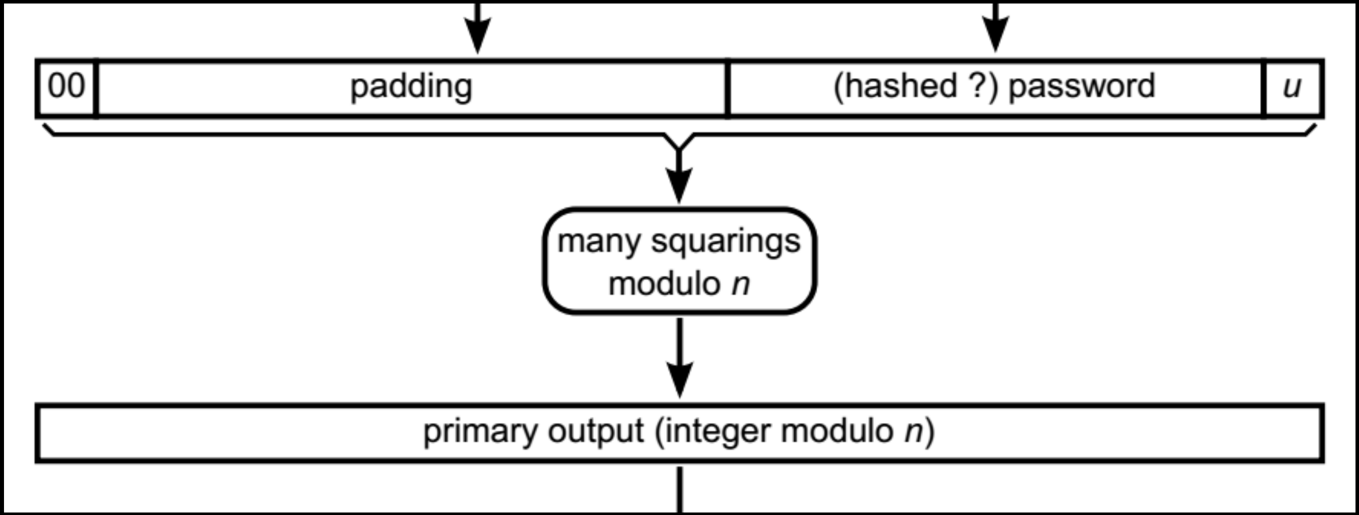


m3lcrypt

WITHDRAWN

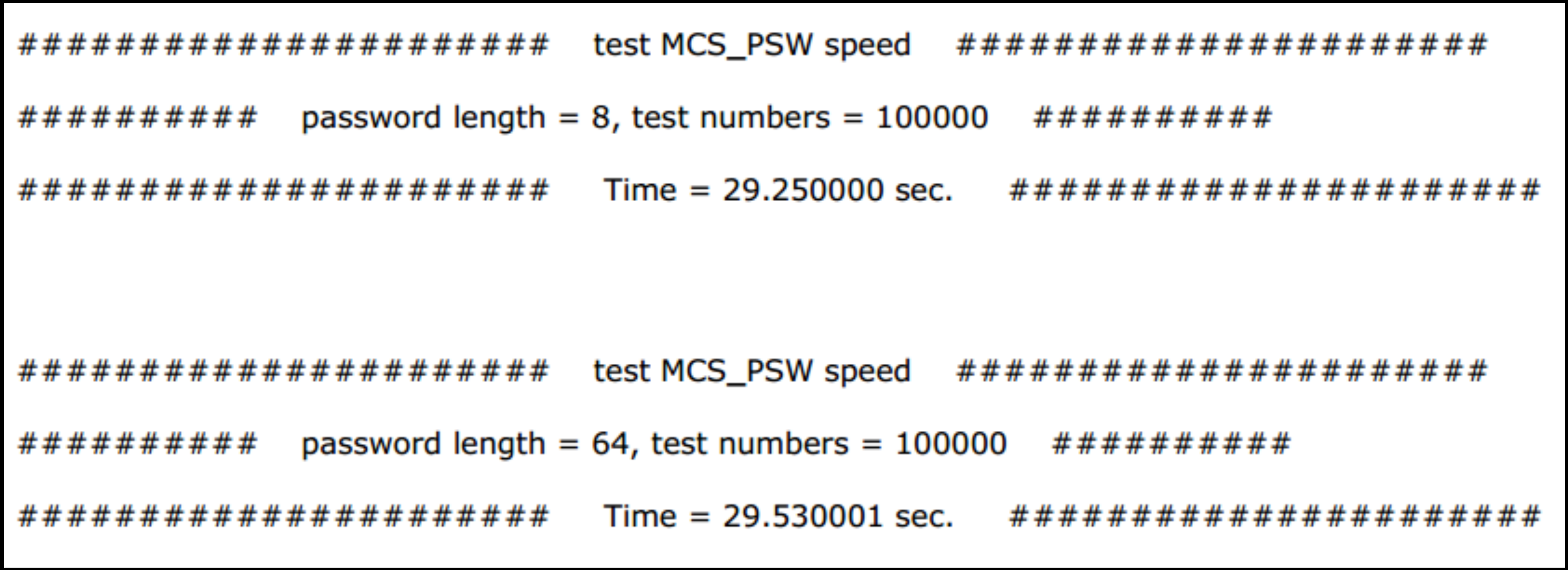
Makwa (Pornin)

- uses **bignum** arithmetic (modular squarings)
- uses HMAC_DRBG
- supports **delegation** to untrusted systems
- supports **password escrow**, **hash upgrade**


00
padding
(hashed ?) password
u
many squarings modulo *n*
primary output (integer modulo *n*)

MCS_PHS (Maslennikov)

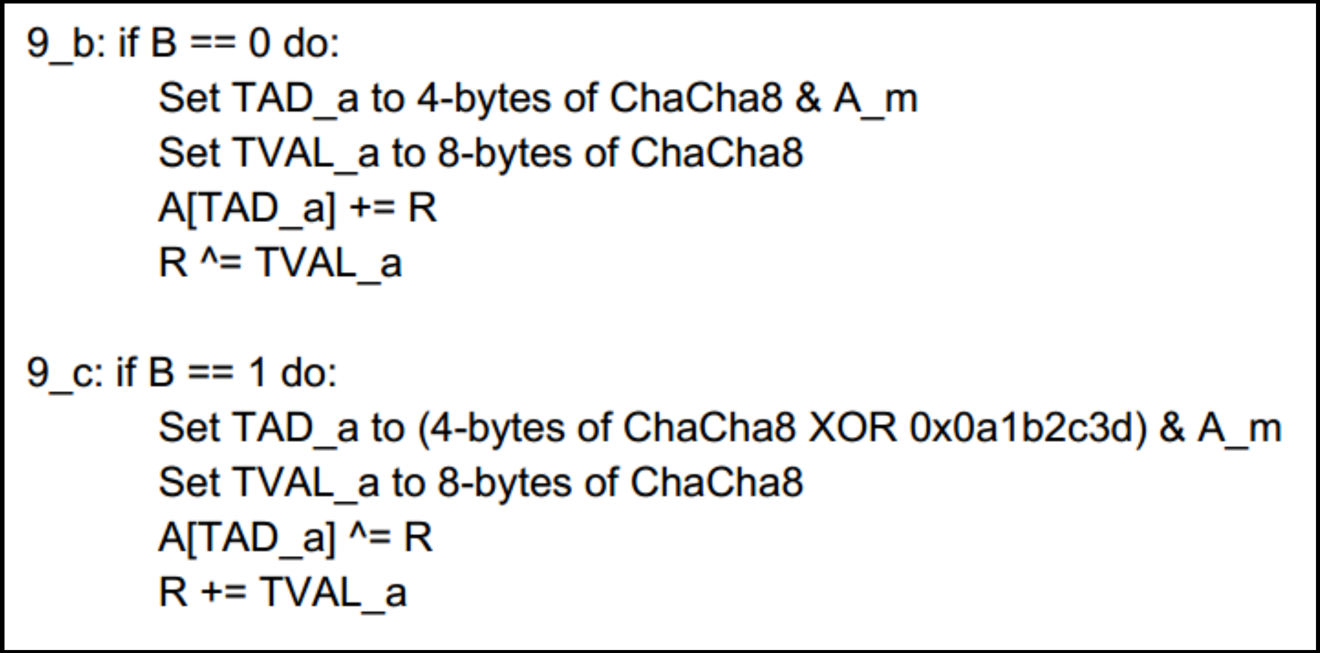
- uses PBKDF2 with MCS_SHA8
- from the MCSSHA* SHA-3 submission…
- simple algorithm: a tweaked PBKDF2



```
######################    test MCS_PSW speed    ######################
##########    password length = 8, test numbers = 100000    ##########
######################     Time = 29.250000 sec.     ######################


######################    test MCS_PSW speed    ######################
##########    password length = 64, test numbers = 100000    ##########
######################     Time = 29.530001 sec.     ######################
```

Omega Crypt (Enright)

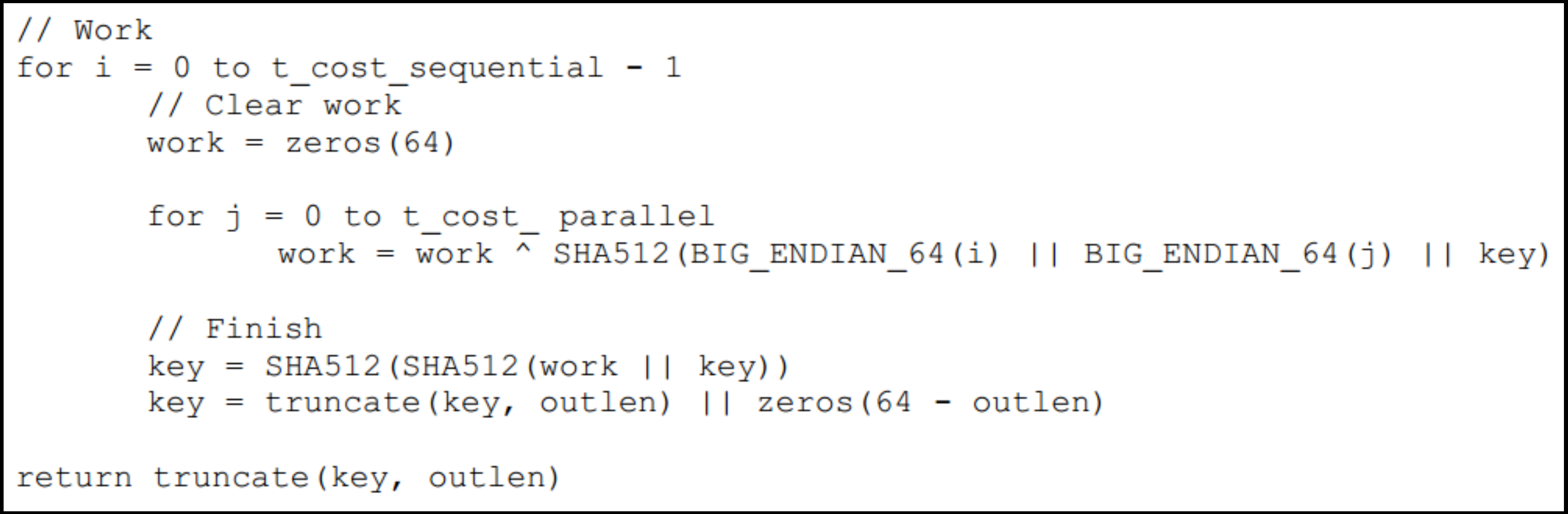
- uses **ChaCha** and **CubeHash** (SIMD-friendly)
- data-dependent branchings…
- … yet timing attack mitigation



```
9_b: if B == 0 do:
        Set TAD_a to 4-bytes of ChaCha8 & A_m
        Set TVAL_a to 8-bytes of ChaCha8
        A[TAD_a] += R
        R ^= TVAL_a

9_c: if B == 1 do:
        Set TAD_a to (4-bytes of ChaCha8 XOR 0x0a1b2c3d) & A_m
        Set TVAL_a to 8-bytes of ChaCha8
        A[TAD_a] ^= R
        R += TVAL_a
```

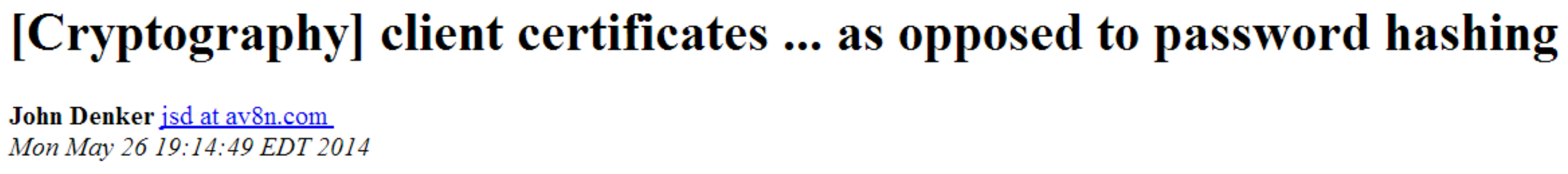
Parallel (Thomas)

- uses **SHA-512**
- **2-dimension** time cost: sequential & parallel
- constant **(low) memory**
- minimalistic and compact design



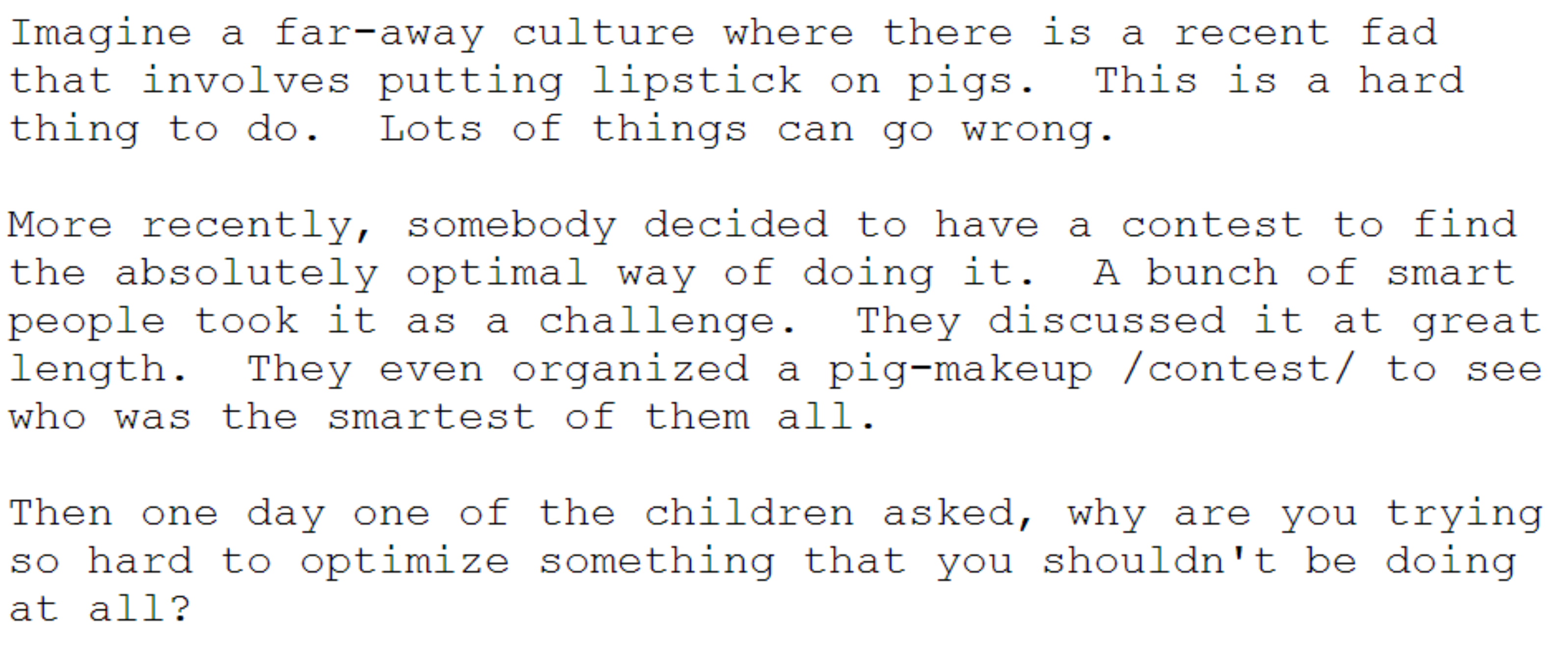
```
// Work
for i = 0 to t_cost_sequential - 1
       // Clear work
       work = zeros(64)

       for j = 0 to t_cost_ parallel
              work = work ^ SHA512(BIG_ENDIAN_64(i) || BIG_ENDIAN_64(j) || key)

       // Finish
       key = SHA512(SHA512(work || key))
       key = truncate(key, outlen) || zeros(64 - outlen)

return truncate(key, outlen)
```

is PHC worthless? :-)



[Cryptography] client certificates ... as opposed to password hashing

John Denker jsd at av8n.com
Mon May 26 19:14:49 EDT 2014



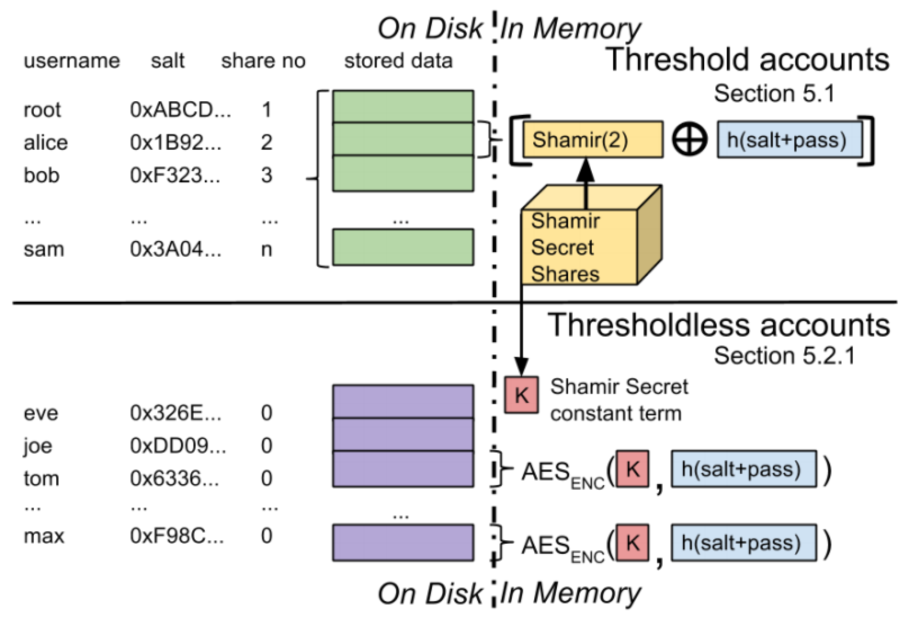
Imagine a far-away culture where there is a recent fad that involves putting lipstick on pigs. This is a hard thing to do. Lots of things can go wrong.

More recently, somebody decided to have a contest to find the absolutely optimal way of doing it. A bunch of smart people took it as a challenge. They discussed it at great length. They even organized a pig-makeup /contest/ to see who was the smartest of them all.

Then one day one of the children asked, why are you trying so hard to optimize something that you shouldn't be doing at all?

PolyPassHash (Cappos, Arias)

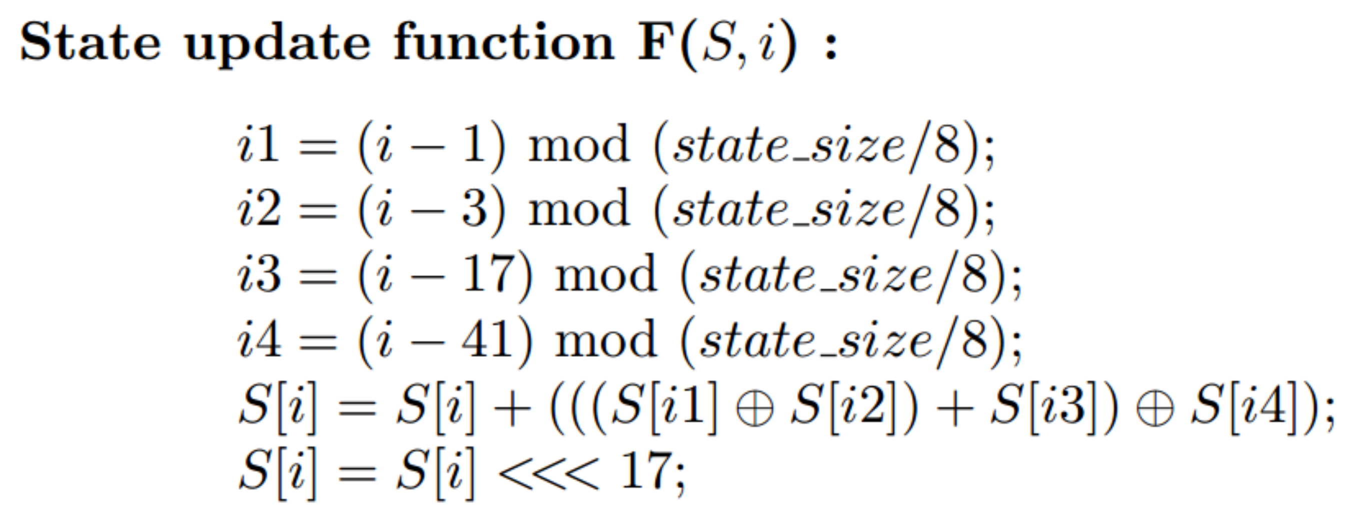
- uses **AES**, **SHA-256**, **SSS**
- threshold of pwds needed to unlock the DB
- only appropriate when **many users**



username	salt	share no	stored data
root	0xABCD...	1	🟩
alice	0x1B92...	2	🟩
bob	0xF323...	3	🟩
...	...	...	...
sam	0x3A04...	n	🟩
eve	0x326E...	0	🟪
joe	0xDD09...	0	🟪
tom	0x6336...	0	🟪
...	...	...	...
max	0xF98C...	0	🟪

POMELO (Wu)

- **no external primitive** (fully original algorithm)
- simple **FSR-like** update functions
- partial mitigation of **cache-timing** attacks
- **compact** self-contained implementations

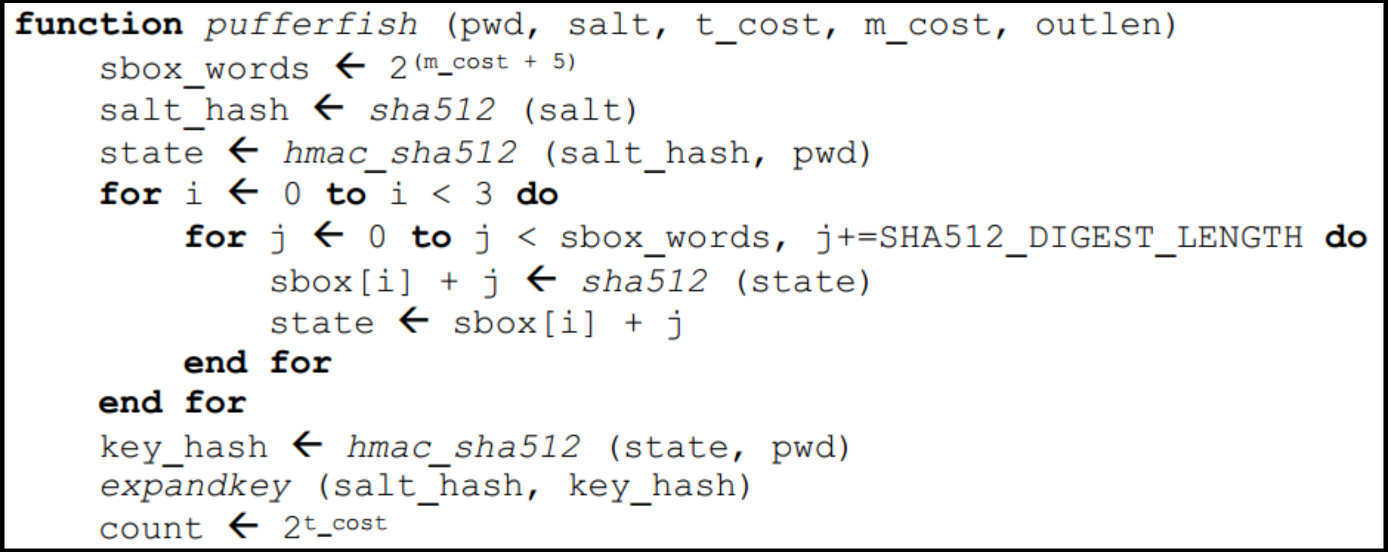


State update function $\mathbf{F}(S, i)$:

```
    $i1 = (i - 1) \bmod (state\_size/8);$
    $i2 = (i - 3) \bmod (state\_size/8);$
    $i3 = (i - 17) \bmod (state\_size/8);$
    $i4 = (i - 41) \bmod (state\_size/8);$
    $S[i] = S[i] + (((S[i1] \oplus S[i2]) + S[i3]) \oplus S[i4]);$
    $S[i] = S[i] <<< 17;$
```

Pufferfish (Gosney)

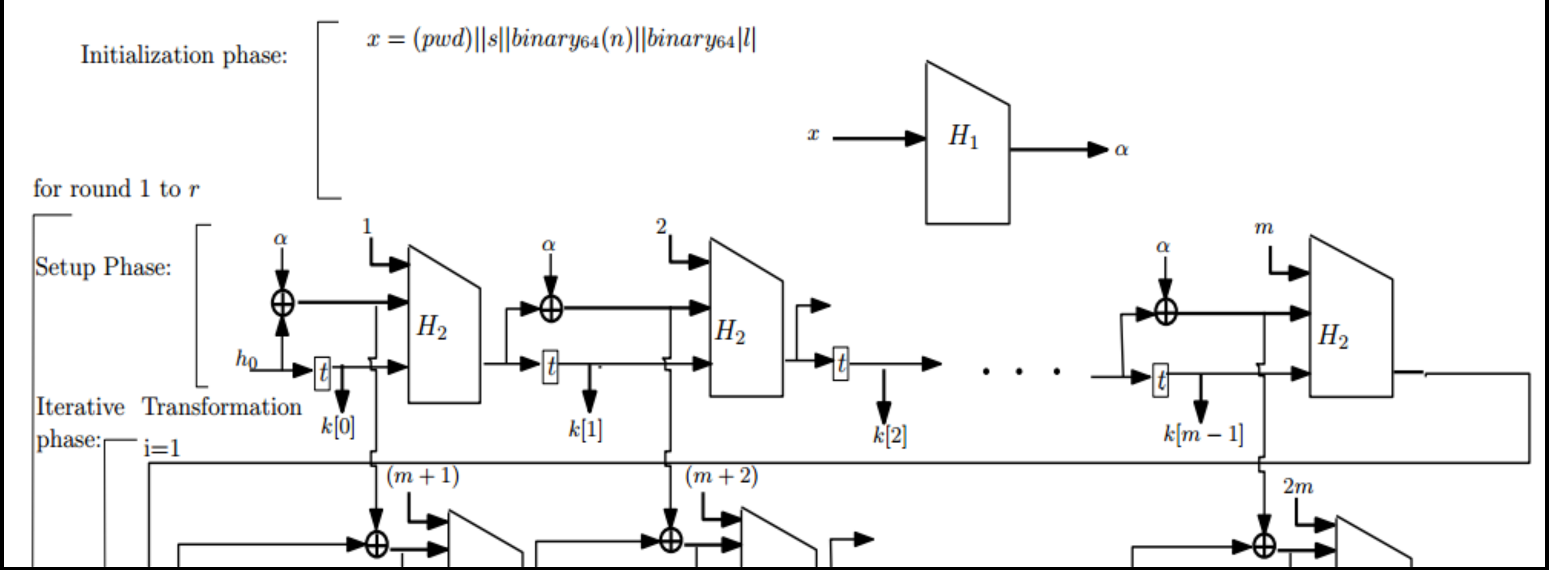
- uses **Blowfish**, **HMAC-SHA-512**
- tweaked Blowfish (pwd-dependent S-boxes, etc.)
- a “modern” bcrypt (64-bit, variable memory)
- JTR patches available



```
function pufferfish (pwd, salt, t_cost, m_cost, outlen)
    sbox_words ← $2^{(m\_cost + 5)}$
    salt_hash ← sha512 (salt)
    state ← hmac_sha512 (salt_hash, pwd)
    for i ← 0 to i < 3 do
        for j ← 0 to j < sbox_words, j+=SHA512_DIGEST_LENGTH do
            sbox[i] + j ← sha512 (state)
            state ← sbox[i] + j
        end for
    end for
    key_hash ← hmac_sha512 (state, pwd)
    expandkey (salt_hash, key_hash)
    count ← $2^{t\_cost}$
```

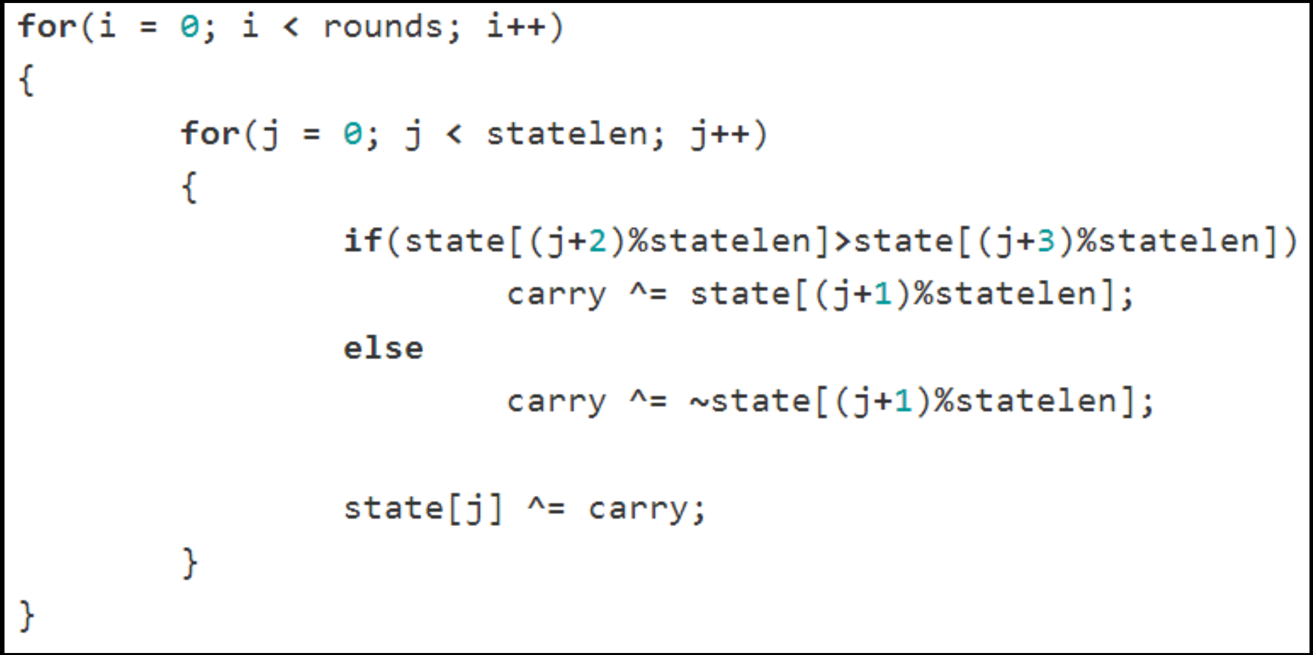
RIG (Chang, Jati, Mishra, Sanadhya)

- uses **BLAKE2b**
- bit-reversal permutation
- mitigation of cache-timing leaks
- supports **server relief** and **hash upgrade**



Schvrch (Vuckovac)

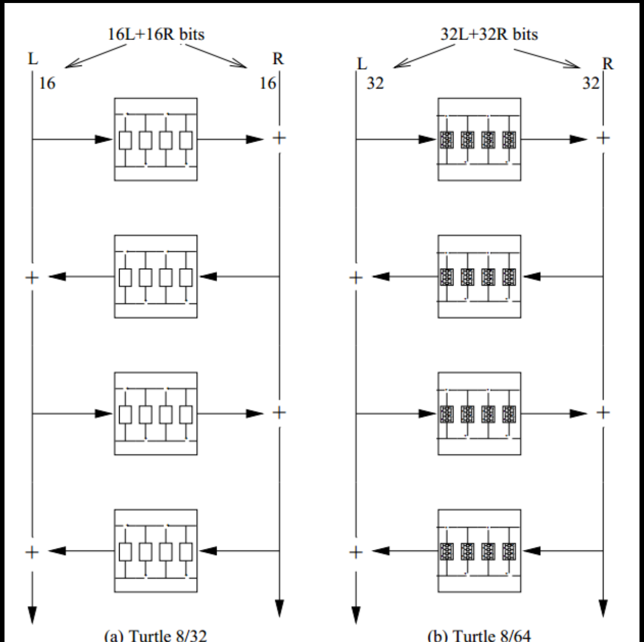
- **no external primitive** (fully original algorithm)
- separate “slow” and “big” computations
- extra “round” parameter for more slow down



```
for(i = 0; i < rounds; i++)
{
        for(j = 0; j < statelen; j++)
        {
                if(state[(j+2)%statelen]>state[(j+3)%statelen])
                        carry ^= state[(j+1)%statelen];
                else
                        carry ^= ~state[(j+1)%statelen];

                state[j] ^= carry;
        }
}
```

Tortuga (Sch)

- uses **Turtle** (Blaze, 1996) as permutation
- keyed sponge structure (absorb/squeeze)
- original and simple construction



(a) Turtle 8/32

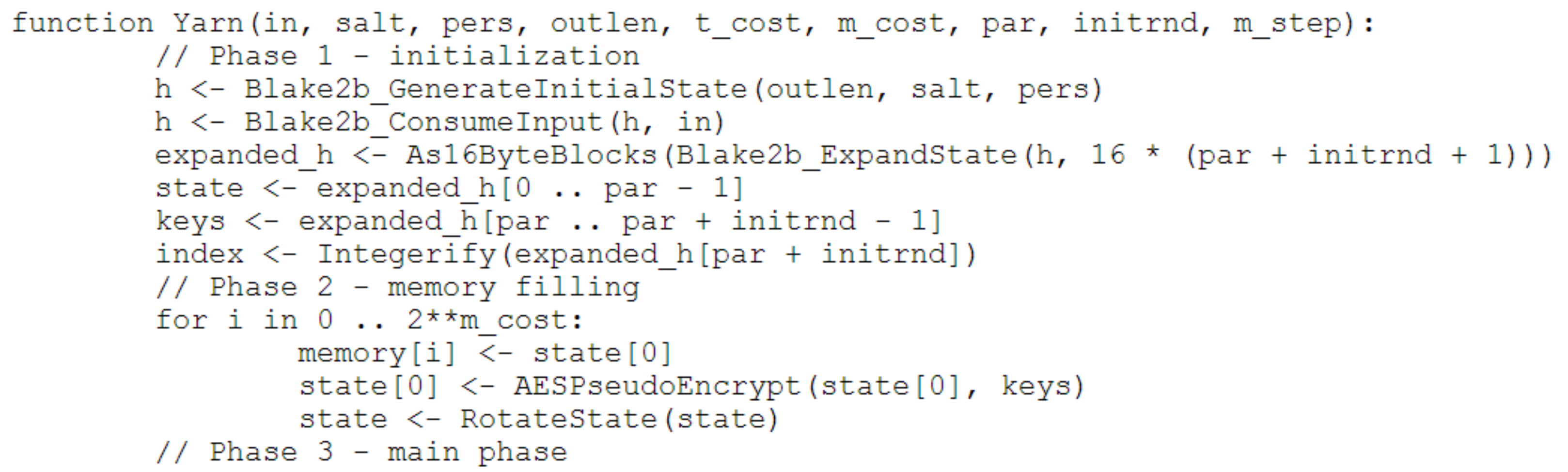
(b) Turtle 8/64

TwoCats (Cox)

- uses **BLAKE2s | BLAKE2b | SHA-256 | SHA-512**
- uses integer **multiplications** (fast on CPUs)
- tweakable thread- and instruction-level **//ism**
- supports **server relief** and **hash upgrade**

Yarn (Capun)

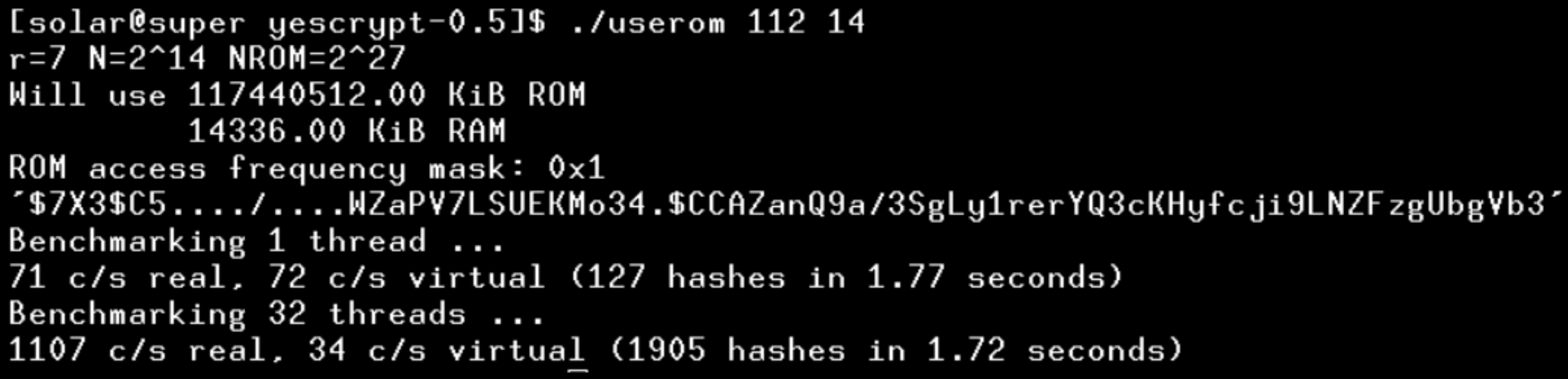
- uses **AES** round and **BLAKE2b**
- parallelism parameterizable
- 3 “time” parameters for distinct resources
- simple and compact design



```
function Yarn(in, salt, pers, outlen, t_cost, m_cost, par, initrnd, m_step):
        // Phase 1 - initialization
        h <- Blake2b_GenerateInitialState(outlen, salt, pers)
        h <- Blake2b_ConsumeInput(h, in)
        expanded_h <- As16ByteBlocks(Blake2b_ExpandState(h, 16 * (par + initrnd + 1)))
        state <- expanded_h[0 .. par - 1]
        keys <- expanded_h[par .. par + initrnd - 1]
        index <- Integerify(expanded_h[par + initrnd])
        // Phase 2 - memory filling
        for i in 0 .. 2**m_cost:
                memory[i] <- state[0]
                state[0] <- AESPseudoEncrypt(state[0], keys)
                state <- RotateState(state)
        // Phase 3 - main phase
```

yescrypt (Peslyak a.k.a. Solar Designer)

- uses **scrypt** with optional tweaks (via bit flags)
- optional: local **ROM**, Salsa20 replacement
- more **parallelism options** (thread and inst. level)
- supports **server relief**



```
[solar@super yescrypt-0.5]$ ./userom 112 14
r=7 N=2^14 NROM=2^27
Will use 117440512.00 KiB ROM
         14336.00 KiB RAM
ROM access frequency mask: 0x1
'$7X3$C5..../....WZaPV7LSUEKMo34.$CCAZanQ9a/3SgLy1rerYQ3cKHyfcji9LNZFzgUbgVb3'
Benchmarking 1 thread ...
71 c/s real, 72 c/s virtual (127 hashes in 1.77 seconds)
Benchmarking 32 threads ...
1107 c/s real, 34 c/s virtual (1905 hashes in 1.72 seconds)
```

NOT
THE
END

3. next steps, and how to contribute

in Q3 2014, we’ll select the **finalists**

(probably between 5 and 10)

in Q2 2015, we’ll select the **winners**, expected to become *de facto standards*

some panel members submitted:
we’ll avoid **conflicts of interest**

evaluation criteria

security (pseudorandomness, etc.)

efficiency ratio (e.g. CPU vs GPU)

simplicity (#LoCs, dependencies, etc.)

extra functionalities

target application

etc.

transparency

we’ll try to have **public discussions** as much as possible

a **final report** will be published, justifying our choices

we need

reviews of the implementations
https://github.com/bsdphk/PHC/

third-party implementations
(to check consistency with the specs, etc.)

cryptanalysis
(memory bypass, side-channel attacks, etc.)

any comment or suggestion to improve

Thank you!