The hash function family LAKE

Jean-Philippe Aumasson, Willi Meier, Raphael C.-W. Phan

University of Applied Sciences Northwestern Switzerland
School of Engineering

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Hash functions at FSE

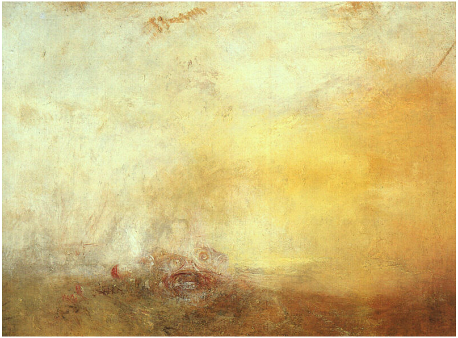
FSE **08**: LAKE

FSE **07**: Grindahl
$\rightarrow$ broken (AC 07)

FSE **06**: FORK-256
$\rightarrow$ broken (FSE 07)

FSE **05**: SMASH
$\rightarrow$ broken (SAC 05)

DESIGN OF LAKE



Overview

- **Family** = LAKE-256 + LAKE-512 + truncated variants
- HAIFA as iterated mode
- Built-in randomized hashing

Key ideas

- Local "wide-pipe" in the compression function
- Multiple levels of feedforward
- Highly modular structure

HAIFA

$\approx$ Merkle-Damgård with salt and dithering [Biham-Dunkelman 06]

- **Effective initial value** is

$$H_0 = C(\text{digest bitsize}, IV, 0, 0)$$

- **Compression function** computes

$$H_i = C(H_{i-1}, M_i, \text{salt}, \#\text{bits hashed so far})$$

- **Padding** is

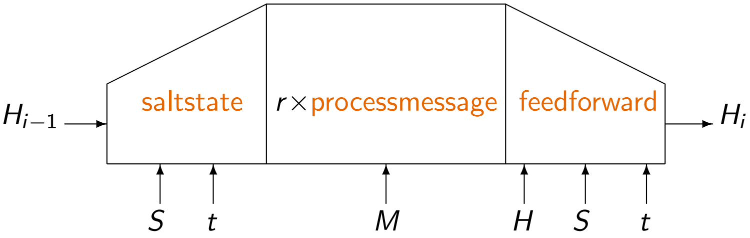
$$1\|0\ldots0\|\text{message bitsize}\|\text{digest bitsize}$$

Side advantages over MD

- Prevents from fixed-point-based attacks
- Makes “herding attacks” harder

LAKE's compression function

Input: 8-word chain value H, 16-word message block M, 4-word salt S, 2-word index t.



- saltstate **stretches** the chain value to 16 words
- processmessage **transforms** the state bijectively
- feedforward **shrinks back** with dependence on H, S and t

The saltstate function

Initialization of the 16-word local chain value L.

input $H_0 \dots H_7, \quad S_0 \dots S_3, \quad t_0 t_1$

1. **for** $i = 0, \dots, 7$ **do**
   $L_i \leftarrow H_i$
2. $L_8 \leftarrow \mathrm{g}(H_0, S_0 \oplus t_0, C_8, 0)$
3. $L_9 \leftarrow \mathrm{g}(H_1, S_1 \oplus t_1, C_9, 0)$
4. **for** $i = 10, \dots, 15$ **do**
   $L_i \leftarrow \mathrm{g}(H_i, S_i, C_i, 0)$

output $L_0 \dots L_{15}$

- Injective mapping
- Uses 32-bit constants $C_8, \dots, C_{15}$

The processmessage function

Message-dependent bijective transform of L.

input $L_0 \dots L_{15}, \quad M_0 \dots M_{15}, \quad \sigma$

1. $F \leftarrow L$
2. **for** $i = 0, \dots, 15$ **do**
   $L_i \leftarrow \mathrm{f}(L_{i-1}, L_i, M_{\sigma(i)}, C_i)$
3. **for** $i = 0, \dots, 15$ **do**
   $L_i \leftarrow \mathrm{g}(L_{i-1}, L_i, F_i, L_{i+1})$

output $L = L_0 \dots L_{15}$

- 8 rounds in LAKE-256, 10 rounds in LAKE-512
- Uses a permutation σ and constants $C_0, \dots, C_{15}$

The feedforward function

Compression of the final L to the new global chain value.

---

input $L_0 \ldots L_{15}, \quad H_0 \ldots H_7, \quad S = S_0 \ldots S_3, \quad t_0 t_1$

1. $H_0 \leftarrow \mathrm{f}(L_0, L_8, S_0 \oplus t_0, H_0)$
2. $H_1 \leftarrow \mathrm{f}(L_1, L_9, S_1 \oplus t_1, H_1)$
3. **for** $i = 2, \ldots, 7$ **do**

   $H_i \leftarrow \mathrm{f}(L_i, L_{i+8}, S_i, H_i)$

output $H_0 \ldots H_7$

- 14 words are fedforward
- Parallelizable into 8 branches

The f function

For LAKE-256:

$$\begin{aligned}\mathsf{f}(a,b,c,d) &= \left[a+(b\vee C_0)\right]+\left(\left[c+(a\wedge C_1)\right]\ggg 7\right)\\ &+ \left(\left[b+(c\oplus d)\right]\ggg 13\right)\end{aligned}$$

- Used in the round function and for global feedforward
- Fast and constant-time operators
- Fast diffusion of changes accross words
- Double input of a, b, c limits absorption by $\vee$ and $\wedge$

The g function

For LAKE-256:

$$\mathrm{g}(a, b, c, d) \quad = \quad \left[(a + b) \ggg 1\right] \oplus (c + d)$$

- Used in the round function for local feedforward
- Very fast, parallelizable
- Basic diffusion of changes
- 1-bit rotation breaks up the byte structure; faster than multibit rotation on some CPU's

Parameters choice

- Bitsizes of digest/message to suit standard API's
- Conservative round numbers (8, 10)
- 128-bit salt (resp. 256) seems sufficient
- 64-bit index (resp. 128) seems sufficient

SECURITY COUNTERMEASURES

Against side-channel attacks

To prevent from:

- Timing attacks
- Power attacks

Countermeasures:

- No S-boxes (risk of cache attacks)
- Constant-time operators ($+, \oplus, \vee, \wedge, \ggg k$)
- Constant-distance rotations
- No (input-dependent) branchings
- No (input-dependent) loads/stores' addresses

Against conventional attacks

- Wide-pipe makes local collisions impossible
- Feedforwards: inversion resistance and complex structure
- Modular structure facilitates analysis
- No trivial fixed-points

Obstacles to differential analysis

- No shift register, to complicate "perturb-and-correct"
- Linear approximations of f and g made difficult
- High number of message inputs: 128 vs. 64 in SHA-256
- Flow dependence

Attacking LAKE

Best attacks known:

- One-round collisions with distinct salts or IV's
- One-round low-weight differential
- Two-round statistical distinguisher

Conjectured:

- LAKE-256 and LAKE-512 preimage and collision resistant
- Salt-indexed function families pseudorandom, unpredictable

Attacking LAKE

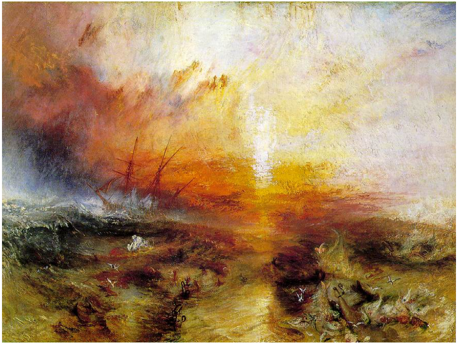
Multiple attack scenarios:

- Chosen/fixed salt/IV attacks,
- Compression function with free index
- Fixed-points/collisions for processmessage

Consider simplified versions:

- Reduce the number of rounds
- Replace f by g
- Change rotation distances
- Use constant constants $C_0 = \cdots = C_{15}$
- Use only the trivial permutation

PERFORMANCE



Algorithmic complexities

LAKE-256 vs. SHA-256

Arithmetic operations:

- 1908 vs. 2232 in total
- 952 vs. 600 integer additions
- 276 vs. 640 XOR's
- 136 vs. 320 AND's
- 136 vs. 0 OR's
- 408 vs. 576 rotations
- 0 vs. 96 shifts

Memory

LAKE-256 vs. SHA-256

Memory (bytes):

- 64 vs. 256 for constants
- 128 vs. 224 for local variables

Benchmarks

LAKE-256 vs. SHA-256

"Moderately" optimized C code for both, gcc 4.1.2, Linux 2.6.19

Estimates of the median cycle count for the compression function:

- Athlon 800 MHz: 2700 vs. 3000 (42 vs. 50 cycles/byte)
- Pentium 4 1500 MHz: 3600 vs. 4000 (56 vs. 63 cycles/byte)
- Pentium 4 2400 MHz: 3300 vs. 3900 (52 vs. 61 cycles/byte)

QUESTIONS

FAQ

Will you submit LAKE to NIST?
$\rightarrow$ *We may submit something based on.*

What about hardware efficiency?
$\rightarrow$ *Implementation is in progress.*

Why an explicit salt when exist generic methods (IV, RMX)?
$\rightarrow$ *To avoid weak home-brewed modes and encourage the use of randomized hashing.*

Where can I get a source code of LAKE?
$\rightarrow$ *Email me.*

The hash function family LAKE

Jean-Philippe Aumasson, Willi Meier, Raphael C.-W. Phan

University of Applied Sciences Northwestern Switzerland
School of Engineering

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE