Ext3 HTree Hashing Details by Alex Meyer

Version	Hash		Since
0	Legacy	(signed)	ext2
1	Half-MD4	(signed)	ext3
2	TEA	(signed)	ext3
3	Legacy	(unsigned)	ext4
4	Half-MD4	(unsigned)	ext4
5	TEA	(unsigned)	ext4
6	SipHash-2-4 (only with fscrypt)		ext4

The hash seed is global, and is stored in the superblock as `s_hash_seed`.
(This is a questionable design decision; it should be randomized on a per-directory basis to reduce the risk of malicious users constructing hash collisions.)

Ext4 documentation on Hash Tree directories

Ext2 Legacy Hash Algorithm

- All operations are done on unsigned 32 bit integers; no operations will wrap.
- `INIT0` = `0x12A3FE2D` (composite)
- `INIT1` = `0x37ABE8F9` (prime)
- `FACTOR` = `0x006D22F5` (prime)

HashLegacy (input: `[u8]`)

`acc0`, `acc1` ← `INIT0`, `INIT1`

for each `byte` **in** input **do**

  `tmp` ← `acc1` + (`acc0` ^ (`byte` * `FACTOR`)) // if signed, sign extend `byte` to 32 bits

  if `tmp` $\geq 2^{31}$ **then**

    `tmp` ← (`tmp` mod 2^{31}) + 1

  `acc0`, `acc1` ← `tmp`, `acc0`

return 2 * `acc0`

Ext2 Hashbuf Encoding

$$\begin{aligned}
\textsc{acc}(v_0, v_1, v_2, v_3) &= (((((v_0 \texttt{ << } 8) + v_1) \texttt{ << } 8) + v_2) \texttt{ << } 8) + v_3 \\
\textsc{acc-part}(p, v_0, \ldots) &= \textsc{acc-part}((p \texttt{ << } 8) + v_0, \ldots) \\
\textsc{acc-part}(p) &= p
\end{aligned}$$

Addition is wrapping and values v_i are optionally sign extended to 32 bits. (In the unsigned variant, ACC is equivalent to converting the bytes to a big-endian integer.)

ENCODE (input: [u8; $0 \le \text{len}(\texttt{input}) \le 4 * \text{len}(\texttt{output})$]) → `output`: [u32]

let `padding` ← $(\text{len}(\texttt{input}) \bmod 256)$ repeated for each byte of a `u32`

for each full 4-byte chunk and its index i **in** `input` **do**

  `output`$[i]$ ← ACC(chunk)

`output`$[\text{last} + 1]$ ← ACC-PART(`padding`, `input`$[\text{rest}..]$)

fill `output`$\left[\left\lceil \frac{\text{len}(\texttt{input})}{4} \right\rceil ..\right]$ with `padding`

return `output`

TEA Encryption Algorithm

- All operations are wrapping, and done on unsigned 32 bit integers.
- `DELTA = 0x9E3779B9` ($= \left\lfloor \frac{1}{\varphi} * 2^{32} \right\rfloor$)
- `ROUNDS = 32`

ENCRYPT (values $(\texttt{v0}, \texttt{v1})$, keys $(\texttt{k0}, \texttt{k1}, \texttt{k2}, \texttt{k3})$)

for each $\texttt{i} = 0, \ldots, \texttt{ROUNDS} - 1$ **do**

$\texttt{v0} \leftarrow \texttt{v0} + ((\texttt{v1} \ll 4) + \texttt{k0}) \mathbin{\hat{}} (\texttt{v1} + \texttt{i} * \texttt{DELTA}) \mathbin{\hat{}} (\texttt{v1} \gg 5) + \texttt{k1}$

$\texttt{v1} \leftarrow \texttt{v1} + ((\texttt{v0} \ll 4) + \texttt{k2}) \mathbin{\hat{}} (\texttt{v0} + \texttt{i} * \texttt{DELTA}) \mathbin{\hat{}} (\texttt{v0} \gg 5) + \texttt{k3}$

return (v_0, v_1)

DECRYPT (values $(\texttt{v0}, \texttt{v1})$, keys $(\texttt{k0}, \texttt{k1}, \texttt{k2}, \texttt{k3})$)

for each $\texttt{i} = \texttt{ROUNDS} - 1, \ldots, 0$ **do**

$\texttt{v1} \leftarrow \texttt{v1} - ((\texttt{v0} \ll 4) + \texttt{k2}) \mathbin{\hat{}} (\texttt{v0} + \texttt{i} * \texttt{DELTA}) \mathbin{\hat{}} (\texttt{v0} \gg 5) + \texttt{k3}$

$\texttt{v0} \leftarrow \texttt{v0} - ((\texttt{v1} \ll 4) + \texttt{k0}) \mathbin{\hat{}} (\texttt{v1} + \texttt{i} * \texttt{DELTA}) \mathbin{\hat{}} (\texttt{v1} \gg 5) + \texttt{k1}$

return (v_0, v_1)

Half-MD4 Transform

- Integer operations are unsigned and wrap modulo 2^{32}.
- $\lll$ is bitwise rotation.
- `K1 = 0x00000000`
- `K2 = 0x5A827999` ($= \left\lfloor \sqrt{2} * 2^{30} \right\rfloor$)
- `K3 = 0x6ED9EBA1` ($= \left\lfloor \sqrt{3} * 2^{30} \right\rfloor$)

$$F(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$$
$$G(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$$
$$H(x, y, z) = x \oplus y \oplus z$$

- The Half MD4 transform used in Ext differs from the full MD4 transform in two ways: the input size (8 words instead of 16 words) and the input indexing;

Round	Half	Full
1	$4i + j$	$4i + j$
2	$1 - i + 2j$	$i + 4j$
3	$-2i + [3, 7, 2, 6][j]$	$[0, 2, 1, 3][i] + 4 * [0, 2, 1, 3][j]$

HalfMD4Transform ((A, B, C, D): [u32; 4], input (X): [u32; 8])

At, Bt, Ct, Dt ← A, B, C, D

for each $i = 0, ..., |X|/4$ **do** // Round 1

- $A \leftarrow (A + F(B, C, D) + X[0 + 4i] + \texttt{K1}) \lll 3$
- $D \leftarrow (D + F(A, B, C) + X[1 + 4i] + \texttt{K1}) \lll 7$
- $C \leftarrow (C + F(D, A, B) + X[2 + 4i] + \texttt{K1}) \lll 11$
- $B \leftarrow (B + F(C, D, A) + X[3 + 4i] + \texttt{K1}) \lll 19$

for each $i = 0, ..., |X|/4$ **do** // Round 2

- $A \leftarrow (A + G(B, C, D) + X[1 - i] + \texttt{K2}) \lll 3$
- $D \leftarrow (D + G(A, B, C) + X[3 - i] + \texttt{K2}) \lll 5$
- $C \leftarrow (C + G(D, A, B) + X[5 - i] + \texttt{K2}) \lll 9$
- $B \leftarrow (B + G(C, D, A) + X[7 - i] + \texttt{K2}) \lll 13$

for each $i = 0, ..., |X|/4$ **do** // Round 3

- $A \leftarrow (A + H(B, C, D) + X[3 - 2i] + \texttt{K3}) \lll 3$
- $D \leftarrow (D + H(A, B, C) + X[7 - 2i] + \texttt{K3}) \lll 9$
- $C \leftarrow (C + H(D, A, B) + X[2 - 2i] + \texttt{K3}) \lll 11$
- $B \leftarrow (B + H(C, D, A) + X[6 - 2i] + \texttt{K3}) \lll 15$

return At + A, Bt + B, Ct + C, Dt + D

Ext2 TEA and Half-MD4 Hashes

- The hashes are seeded with four 32 bit usigned integers.
- If unset, the seed defaults to [`0x67452301`, `0xEFCDAB89`, `0x98BADCFE`, `0x10325476`].
- The unsigned variants use the (faster) unsigned version of HashbufEncode.

HashTEA (seed: `[u32; 4]`, input: `[u8]`)

```
v0, v1 ← seed[0], seed[1]
for each 16 byte chunk in input do
    k0, k1, k2, k3 ← HashbufEncode(chunk)
    v0, v1 ← TEAEncrypt([v0, v1], [k0, k1, k2, k3])
return v0, v1
```

HashMD4Half (seed: `[u32; 4]`, input: `[u8]`)

```
state ← seed
for each 32 byte chunk in input do
    words ← HashbufEncode(chunk)
    state ← HalfMD4Transform(state, words)
return state[1], state[2]
```