

Nakamichi's keyshrinker **DoubleDeuceAES_YMM** - hashing (using AES) smallkeys (1..64..) down to 16 bytes

[illegible]

```

c2YMM = _mm256_unpackhi_epi8 (a0YMM, a2YMM);
tmp0YMM = _mm256_shuffle_epi8 (a0YMM, PartialInterleavingMask1YMM);
tmp2YMM = _mm256_shuffle_epi8 (a2YMM, PartialInterleavingMask3YMM);
c0YMM = _mm256_or_si256 (tmp0YMM, tmp2YMM);
tmp0YMM = _mm256_shuffle_epi8 (a0YMM, PartialInterleavingMask1YMM);
tmp2YMM = _mm256_shuffle_epi8 (a2YMM, PartialInterleavingMask3YMM);
c2YMM = _mm256_or_si256 (tmp0YMM, tmp2YMM);
//tmp0 = _mm_shuffle_epi8 (b0, PartialInterleavingMask1);
//tmp1 = _mm_shuffle_epi8 (b0, PartialInterleavingMask2);
//tmp2 = _mm_shuffle_epi8 (b2, PartialInterleavingMask3);
//tmp3 = _mm_shuffle_epi8 (b2, PartialInterleavingMask4);
//d0 = _mm_or_si128 (tmp0, tmp2);
//d1 = _mm_or_si128 (tmp1, tmp3);
//tmp0 = _mm_shuffle_epi8 (b1, PartialInterleavingMask1);
//tmp1 = _mm_shuffle_epi8 (b1, PartialInterleavingMask2);
//tmp2 = _mm_shuffle_epi8 (b3, PartialInterleavingMask3);
//tmp3 = _mm_shuffle_epi8 (b3, PartialInterleavingMask4);
//d2 = _mm_or_si128 (tmp0, tmp2);
//d3 = _mm_or_si128 (tmp1, tmp3);
// [[ Next 6 lines are identical to simply REVERSE C vector - which is in 2 lines
/*
tmp0YMM = _mm256_shuffle_epi8 (b0YMM, PartialInterleavingMask1YMM);
tmp2YMM = _mm256_shuffle_epi8 (b2YMM, PartialInterleavingMask3YMM);
d0YMM = _mm256_or_si256 (tmp0YMM, tmp2YMM);
tmp0YMM = _mm256_shuffle_epi8 (b0YMM, PartialInterleavingMask2YMM);
tmp2YMM = _mm256_shuffle_epi8 (b2YMM, PartialInterleavingMask4YMM);
d2YMM = _mm256_or_si256 (tmp0YMM, tmp2YMM);
*/
// ]]] Next 6 lines are identical to simply REVERSE C vector - which is in 2 lines, but I don't have it, so {{{
d0YMM = _mm256_unpacklo_epi8 (b0YMM, b2YMM);
d2YMM = _mm256_unpackhi_epi8 (b0YMM, b2YMM);
// }}}
//In YMM should swap c1 and c2, also d1 and d2
//hashA = _mm_aesenc_si128(hashA, a0);
hashA = _mm_aesenc_si128(hashA, *((_m128i *)(&a0YMM)));
//hashB = _mm_aesenc_si128(hashB, b0);
hashB = _mm_aesenc_si128(hashB, *((_m128i *)(&b0YMM)));
//hashB = _mm_aesenc_si128(hashB, *((_m128i *)(&b0YMM)+1));
//hashC = _mm_aesenc_si128(hashC, c0);
hashC = _mm_aesenc_si128(hashC, *((_m128i *)(&c0YMM)));
//hashD = _mm_aesenc_si128(hashD, d0);
hashD = _mm_aesenc_si128(hashD, *((_m128i *)(&d0YMM)));
//hashA = _mm_aesenc_si128(hashA, a1);
hashA = _mm_aesenc_si128(hashA, *((_m128i *)(&a0YMM)+1));
//hashB = _mm_aesenc_si128(hashB, b1);
hashB = _mm_aesenc_si128(hashB, *((_m128i *)(&b0YMM)+1));
//hashB = _mm_aesenc_si128(hashB, *((_m128i *)(&b0YMM)));
//hashC = _mm_aesenc_si128(hashC, c1);
hashC = _mm_aesenc_si128(hashC, *((_m128i *)(&c2YMM)));
//hashD = _mm_aesenc_si128(hashD, d1);
hashD = _mm_aesenc_si128(hashD, *((_m128i *)(&d2YMM)));
//hashA = _mm_aesenc_si128(hashA, a2);
hashA = _mm_aesenc_si128(hashA, *((_m128i *)(&a2YMM)));
//hashB = _mm_aesenc_si128(hashB, b2);
hashB = _mm_aesenc_si128(hashB, *((_m128i *)(&b2YMM)));
//hashB = _mm_aesenc_si128(hashB, *((_m128i *)(&b2YMM)+1));
//hashC = _mm_aesenc_si128(hashC, c2);
hashC = _mm_aesenc_si128(hashC, *((_m128i *)(&c0YMM)+1));
//hashD = _mm_aesenc_si128(hashD, d2);
hashD = _mm_aesenc_si128(hashD, *((_m128i *)(&d0YMM)+1));
//hashA = _mm_aesenc_si128(hashA, a3);
hashA = _mm_aesenc_si128(hashA, *((_m128i *)(&a2YMM)+1));
//hashB = _mm_aesenc_si128(hashB, b3);
hashB = _mm_aesenc_si128(hashB, *((_m128i *)(&b2YMM)+1));
//hashB = _mm_aesenc_si128(hashB, *((_m128i *)(&b2YMM)));
//hashC = _mm_aesenc_si128(hashC, c3);
hashC = _mm_aesenc_si128(hashC, *((_m128i *)(&c2YMM)+1));
//hashD = _mm_aesenc_si128(hashD, d3);
hashD = _mm_aesenc_si128(hashD, *((_m128i *)(&d2YMM)+1));
hashA = _mm_aesenc_si128(hashA, hashB);
hashA = _mm_aesenc_si128(hashA, hashC);
hashA = _mm_aesenc_si128(hashA, hashD);
length = length - 64;
}
}
ptr128 = (_m128i *)buffer;
if (length >= 16) {
    cycles = length/16;
    for (; cycles--; buffer += 16) {
        AgainstRules = _mm_loadu_si128(ptr128++);
        GumbotronREVER = _mm_shuffle_epi8 (AgainstRules, ReverseMask);
        GumbotronINTER = _mm_shuffle_epi8 (AgainstRules, InterleaveMask);
        GumbotronREVERINTER = _mm_shuffle_epi8 (GumbotronREVER, InterleaveMask);
        hashA = _mm_aesenc_si128(hashA, AgainstRules);
        hashB = _mm_aesenc_si128(hashB, GumbotronREVER);
        hashC = _mm_aesenc_si128(hashC, GumbotronINTER);
        hashD = _mm_aesenc_si128(hashD, GumbotronREVERINTER);
        hashA = _mm_aesenc_si128(hashA, hashB);
        hashA = _mm_aesenc_si128(hashA, hashC);
        hashA = _mm_aesenc_si128(hashA, hashD);
        length = length - 16;
    }
}
// Inhere using Pippin's approach to read past the end ("the dirty" sentinel like style, or more like padding):
if (length & (16-1)) {
    AgainstRules = _mm_loadu_si128(ptr128);
    //AgainstRules = _mm_srli_si128 (AgainstRules, 16-length); // catastrophic error: Intrinsic parameter must be an immediate value
    AgainstRules = _mm_and_si128 (AgainstRules, Gumbotron[length]);
    //Gumbotron = _mm_slli_si128 (Gumbotron, 16-length); // catastrophic error: Intrinsic parameter must be an immediate value
    Gumbotron = _mm_and_si128 (hashB, Gumbotron[length]);
    AgainstRules = _mm_or_si128 (AgainstRules, Gumbotron);
    GumbotronREVER = _mm_shuffle_epi8 (AgainstRules, ReverseMask);
    GumbotronINTER = _mm_shuffle_epi8 (AgainstRules, InterleaveMask);
    GumbotronREVERINTER = _mm_shuffle_epi8 (GumbotronREVER, InterleaveMask);
    hashA = _mm_aesenc_si128(hashA, AgainstRules);
    hashB = _mm_aesenc_si128(hashB, GumbotronREVER);
    hashC = _mm_aesenc_si128(hashC, GumbotronINTER);
    hashD = _mm_aesenc_si128(hashD, GumbotronREVERINTER);
    hashA = _mm_aesenc_si128(hashA, hashB);
    hashA = _mm_aesenc_si128(hashA, hashC);
    hashA = _mm_aesenc_si128(hashA, hashD);
}
SlowCopy128bit( (const char *)(&hashA), (char *)&DDAES[0]);
}

```

```

// Okay, the AVX2 mainloop (the handler of multiples of 64 bytes) code
// is only 40 instructions (icc 19.0.0 -O3 -mavx2):
// https://godbolt.org/z/0aw3ZGcv5
/*
vmovdqu ymm10, YMMWORD PTR [rdi] #94.43
dec rax #89.9
vmovdqu ymm11, YMMWORD PTR [32+rdi] #95.43
vpshufb ymm8, ymm10, ymm0 #101.12
vpshufb ymm7, ymm11, ymm0 #100.12
vpunpcklbw ymm9, ymm10, ymm11 #191.12
vpunpckhbw ymm12, ymm10, ymm11 #192.12
vmovdqu YMMWORD PTR [64+rsp], ymm9 #191.4
vmovdqu YMMWORD PTR [96+rsp], ymm12 #192.4
vpermq ymm14, ymm7, 78 #104.9
add rsi, -64 #494.22
vpermq ymm15, ymm8, 78 #105.9
vpunpcklbw ymm13, ymm14, ymm15 #253.12
vpunpckhbw ymm7, ymm14, ymm15 #254.12
vmovdqu YMMWORD PTR [rsp], ymm14 #104.1
vmovdqu YMMWORD PTR [32+rsp], ymm15 #105.1
vmovdqu YMMWORD PTR [128+rsp], ymm13 #253.4
vmovdqu YMMWORD PTR [160+rsp], ymm7 #254.4
vaesenc xmm1, xmm1, XMMWORD PTR [rdi] #452.12
vaesenc xmm3, xmm3, XMMWORD PTR [rsp] #454.13
vaesenc xmm4, xmm4, XMMWORD PTR [64+rsp] #457.12
vaesenc xmm3, xmm3, XMMWORD PTR [16+rsp] #464.13
vaesenc xmm1, xmm1, XMMWORD PTR [16+rdi] #462.12
vaesenc xmm7, xmm1, XMMWORD PTR [32+rdi] #472.12
vaesenc xmm6, xmm6, XMMWORD PTR [128+rsp] #459.12
vaesenc xmm4, xmm4, XMMWORD PTR [96+rsp] #467.12
vaesenc xmm8, xmm3, XMMWORD PTR [32+rsp] #474.13
vaesenc xmm6, xmm6, XMMWORD PTR [160+rsp] #469.12
vaesenc xmm9, xmm4, XMMWORD PTR [80+rsp] #477.12
vaesenc xmm3, xmm8, XMMWORD PTR [48+rsp] #484.13
vaesenc xmm11, xmm7, XMMWORD PTR [48+rdi] #482.12
add rdi, 64 #89.19
vaesenc xmm10, xmm6, XMMWORD PTR [144+rsp] #479.12
vaesenc xmm4, xmm9, XMMWORD PTR [112+rsp] #487.12
vaesenc xmm12, xmm11, xmm3 #491.12
vaesenc xmm6, xmm10, XMMWORD PTR [176+rsp] #489.12
vaesenc xmm13, xmm12, xmm4 #492.12
vaesenc xmm1, xmm13, xmm6 #493.12
cmp rax, -1 #89.9
jne .B2. # Prob 82% #89.9
*/

```

Nakamichi's keyshrinker **DoubleDeuceAES_XMM** - hashing (using AES) smallkeys (1..64..) down to 16 bytes

```
#include <stdlib.h>
#include <stdint.h> //uint64_t needed
#include <string.h>
#include <mmintrin.h> //SSE4.1 intrinsics
#include <emmintrin.h>
void SlowCopy128bit(const char *SOURCE, char *TARGET) { __mm_storeu_si128((__m128i *) (TARGET), __mm_loadu_si128((const __m128i *) (SOURCE))); }
```

[illegible]

```
static const __m128i *Mumbotron = (__m128i *) VectorsNeedNonVariable1;
//static const uint8_t VectorsNeedNonVariable2[256] __attribute__((aligned(16))) =
static const uint8_t VectorsNeedNonVariable2[256] =
{
```

[illegible]

```
static const __m128i *Jumbotron = (__m128i *) VectorsNeedNonVariable2;
// Written by Sannayce, inspired by J. Andrew Rogers's https://github.com/jandrewrogers/AquaHash/blob/master/aquahash.h
// This hash function serves two ... functions - useful for table lookups and to shrink keys (usually 64...256 bytes in)
// Inhere using (when the key is not a multiple of 16, therefore padding is needed) Pippip's approach to read past the
void DoubleDeuceAES_Gumbotron(const uint8_t *buffer, size_t length) {
```

```

size_t i, Cycles;
__m128i hashA = _mm_set_epi64x(0x6c62272e07bb0142, 0x62b821756295c58d); // 0x6c62272e07bb014262b821756295c58d
__m128i hashB = _mm_set_epi64x(0xdd268dbcaac55036, 0xd298c384c4e576cc); // 0xdd268dbcaac550362d98c384c4e576cc8c
__m128i hashC = _mm_set_epi64x(0x8b1536847b6bb3, 0x1023b4c8cae0535); // 0xdd268dbcaac550362d98c384c4e576cc8c
__m128i hashD = _mm_setzero_si128();
__m128i a0,a1,a2,a3; // Instead of this chunkenization, ZMM houses the 4 XMMs, if there is shuffle across all t
__m128i b0,b1,b2,b3;
__m128i c0,c1,c2,c3;
__m128i d0,d1,d2,d3;

__m128i tmp0,tmp1,tmp2,tmp3;
__m128i ReverseMask = _mm_set_epi8(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15);
__m128i PartialInterleavingMask1 = _mm_set_epi8(0x80,7,0x80,6,0x80,5,0x80,4,0x80,3,0x80,2,0x80,1,0x80,0);
__m128i PartialInterleavingMask2 = _mm_set_epi8(0x80,0x0,0x80,0x0,0x80,0x0,0x80,0x0,0x80,0x0,0x80,0x0,0x80,0x0,0x80,0x0);
__m128i PartialInterleavingMask3 = _mm_set_epi8(7,0x80,6,0x80,5,0x80,4,0x80,3,0x80,2,0x80,1,0x80,0,0x80);
__m128i PartialInterleavingMask4 = _mm_set_epi8(0xf,0x80,0x0,0x80,0xd,0x80,0xc,0x80,0xb,0x80,0xa,0x80,9,0x80,8,0x80);
const __m128i *ptr128a, *ptr128b, *ptr128c, *ptr128d;
__m128i AgainstRules, GumbotronREVER, GumbotronINTER, Gumbotron, GumbotronREVERINTER;
const __m128i *ptr128;
__m128i InterleaveMask = _mm_set_epi8(15,7,14,6,13,5,12,4,11,3,10,2,9,1,8,0);
if (len0 >= 64) {

```

```

cycles = Length/64;
for( ; Cycles--; buffer += 64 ) {
    a0 = __m_loadu_si128((__m128i *) (buffer+0*16));
    a1 = __m_loadu_si128((__m128i *) (buffer+1*16));
    a2 = __m_loadu_si128((__m128i *) (buffer+2*16));
    a3 = __m_loadu_si128((__m128i *) (buffer+3*16));
    b0 = __m_shuffle_epi8 (a3, ReverseMask);
    b1 = __m_shuffle_epi8 (a2, ReverseMask);
    b2 = __m_shuffle_epi8 (a1, ReverseMask);
    b3 = __m_shuffle_epi8 (a0, ReverseMask);
    tmp0 = __m_shuffle_epi8 (a0, PartialInterleavingMask1);
    tmp1 = __m_shuffle_epi8 (a0, PartialInterleavingMask2);
    tmp2 = __m_shuffle_epi8 (a2, PartialInterleavingMask3);
    tmp3 = __m_shuffle_epi8 (a2, PartialInterleavingMask4);
    c0 = __m_or_si128 (tmp0, tmp2);
    c1 = __m_or_si128 (tmp1, tmp3);
    tmp0 = __m_shuffle_epi8 (a1, PartialInterleavingMask1);
    tmp1 = __m_shuffle_epi8 (a1, PartialInterleavingMask2);
    tmp2 = __m_shuffle_epi8 (a3, PartialInterleavingMask3);
    tmp3 = __m_shuffle_epi8 (a3, PartialInterleavingMask4);
    c2 = __m_or_si128 (tmp0, tmp2);
    c3 = __m_or_si128 (tmp1, tmp3);
    tmp0 = __m_shuffle_epi8 (b0, PartialInterleavingMask1);
    tmp1 = __m_shuffle_epi8 (b0, PartialInterleavingMask2);
    tmp2 = __m_shuffle_epi8 (b2, PartialInterleavingMask3);
    tmp3 = __m_shuffle_epi8 (b2, PartialInterleavingMask4);
    d0 = __m_or_si128 (tmp0, tmp2);
    d1 = __m_or_si128 (tmp1, tmp3);
    tmp0 = __m_shuffle_epi8 (b1, PartialInterleavingMask1);
    tmp1 = __m_shuffle_epi8 (b1, PartialInterleavingMask2);
    tmp2 = __m_shuffle_epi8 (b3, PartialInterleavingMask3);
    tmp3 = __m_shuffle_epi8 (b3, PartialInterleavingMask4);
    d2 = __m_or_si128 (tmp0, tmp2);
    d3 = __m_or_si128 (tmp1, tmp3);
    hashA = __m_aesenc_si128(hashA, a0);
    hashB = __m_aesenc_si128(hashB, b0);
    hashC = __m_aesenc_si128(hashC, c0);
}

```

Testset: "A billion Knight-Tours variants (each KT with 256 variants, the KT itself omitted) - each 128 bytes long"

Testfile: **1000000000.KnightTours.txt** (130,000,000,000 bytes)

The name of the game - hashing all lines and taking either 5 bytes or 6,7,8 bytes from the hash.

| Hasher | Collisions within first 5 bytes |
|------------------------|--|
| xxh3_64bits v0.8.0 | 1,000,000,000 - 999,545,727 distinct lines = 454,273 |
| DoubleDeuceAES_128bits | 1,000,000,000 - 999,545,796 distinct lines = 454,204 |

| Hasher | Collisions within first 6 bytes |
|------------------------|--|
| XXH3_64bits v0.8.0 | 1,000,000,000 - 999,998,214 distinct lines = 1,786 |
| DoubleDeuceAES_128bits | 1,000,000,000 - 999,998,213 distinct lines = 1,787 |

| Hasher | Collisions within first 7 bytes |
|------------------------|---|
| xxh3_64bits v0.8.0 | 1,000,000,000 - 999,999,989 distinct lines = 11 |
| DoubleDeuceAES_128bits | 1,000,000,000 - 999,999,994 distinct lines = 6 |

| Hasher | Collisions within first 8 bytes |
|------------------------|--|
| xxh3_64bits v0.8.0 | 1,000,000,000 - 1,000,000,000 distinct lines = 0 |
| DoubleDeuceAES_128bits | 1,000,000,000 - 1,000,000,000 distinct lines = 0 |

This is how the console looks like:

...

```
C:\test\COLLISION_Hashliner>GENERATE_Xmillion_Knight-Tours.bat 1000000000
Generating 1000000000 Knight-Tours and dumping them into file ...
```

```
C:\test\COLLISION_Hashliner>Knight-Tour_FNV1A_YoshimitsuTRIADii_vs_CRC32_TRISMUS.exe a8
1000000000 1>1000000000.KnightTours.txt
```

```
C:\test\COLLISION_Hashliner>bench7.bat 1000000000.KnightTours.txt
C:\test\COLLISION_Hashliner>Hashliner YXH3 dump7bytesthash.exe 1000000000 KnightTours.txt
```

```
C:\test\COLLISION_Hashliner>Hashliner.DPAES_dump7byteshash.exe 10000000000 KnightTours.txt
```

```
C:\test\COLLISION_Hashliner>Sandokan QuickSortExternal Deduplicated 4+GB 64/bit Intel.exe
1>10000000000.KnightTours.txt.DDAES.txt
```

Sandokan_QuickSortExternal_4+GB.r.3+, written by Kaze, using Bill Durango's Quicksort source

```
Size of input file: 16,000,000,000
Counting lines ...
Lines encountered: 1,000,000,000
Longest line (including CR if present): 15
Allocated memory for pointers-to-lines in MB: 7629
Assigning pointers ...
sizeof(int), sizeof(void*): 4, 8
Trying to allocate memory for the file itself in MB: 15258 ... OK! Get on with fast internal
accesses.
```

```

Uploading ...
Sorting 1,000,000,000 Pointers ...
Quicksort (Insertionsort for small blocks) commenced ...
 / RightEnd: 000,328,304,267; NumberofSplittings: 0,114,284,204; Done: 100% ...
NumberofComparisons: 34,310,536,510
The time to sort 1,000,000,000 items via Quicksort+Insertionsort was 2,848,402 clocks.
Performance: 12,045,534 Comparisons_128B_long-Per-Second i.e 24,091,068 RandomReads_128B_long-
Per-Second.

```

Dumping the sorted data (Regime=2)...

\ Done 100% ...
Source: 1,000,000

OK! Incoming and resultant file's sizes match

Dumping the sorted data [deduplicated] ...

Dumped 999,999,989 distinct lines.

Dump time: 460,940 clocks.

Total time: 3,347,265 clocks.
performance: 4,790 bytes/clock

Performance: 4,780 bytes/clock.
Done successfully

6.) test\COLLECTION: Working on...

```
C:\test\COLLISION_Hash1ther>sort /R QuickSortExternal_4GB.distinct.txt  
1-10000000000 knightTours.txt xxh3 7bytes 3orABOVE.txt
```

```
c:\test\SQL\TESTON\work\linen.dia.*7b*
```

```
C:\test\COLLISION_Hash1ther>dir ^/b^
```

```
15-Aug-21 12:28      156 10000000000.KnightTours.txt.DDAES.7bytes.2orABOVE.txt
15-Aug-21 11:32      286 10000000000.KnightTours.txt.xhx3.7bytes.2orABOVE.txt
```

```
C:\test\COLLISION_Hashliner>type 1000000000.KnightTours.txt.xxh3.7bytes.2orABOVE.txt
0.000.002      f84627e722e85e
```

| | |
|-----------|----------------|
| 0,000,002 | f0039d0c4e4fce |
| 0,000,002 | c87c64d97df0e7 |

| | |
|-----------|----------------|
| 0,000,002 | bb4344a5546572 |
| 0,000,002 | af2f628f4b3ffb |
| 0,000,002 | -8-b8675-04610 |

| | |
|-----------|----------------|
| 0,000,002 | a8c08675c94610 |
| 0,000,002 | a742cf83948622 |
| 0,000,002 | 657cb9dff2d962 |

| | |
|-----------|----------------|
| 0,000,002 | 037c03d112d502 |
| 0,000,002 | 436aef7ab54ce7 |
| 0,000,002 | 270fcde0563670 |

```
0,000,002      0b533d70915c51
C:\test\COLLISION Hashliner>
```

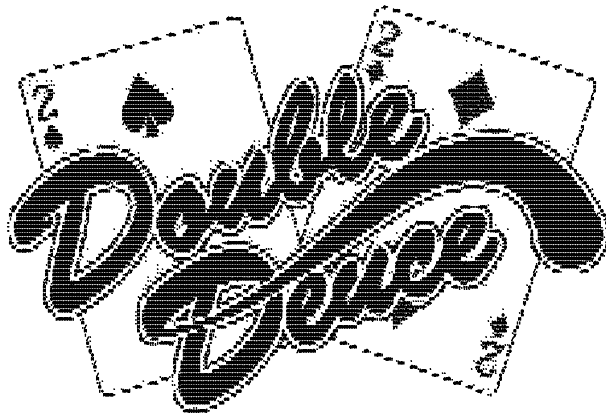
Note: The first column houses how many occurrences of the following hash are there

NOTE: THE FIRST COLUMN SHOWS HOW MANY OCCURRENCES OF THE FOLLOWING HASH ARE THERE:

```

        hashD = _mm_aesenc_si128(hashD, d0);
        hashA = _mm_aesenc_si128(hashA, a1);
        hashB = _mm_aesenc_si128(hashB, b1);
        hashC = _mm_aesenc_si128(hashC, c1);
        hashD = _mm_aesenc_si128(hashD, d1);
        hashA = _mm_aesenc_si128(hashA, a2);
        hashB = _mm_aesenc_si128(hashB, b2);
        hashC = _mm_aesenc_si128(hashC, c2);
        hashD = _mm_aesenc_si128(hashD, d2);
        hashA = _mm_aesenc_si128(hashA, a3);
        hashB = _mm_aesenc_si128(hashB, b3);
        hashC = _mm_aesenc_si128(hashC, c3);
        hashD = _mm_aesenc_si128(hashD, d3);
        hashA = _mm_aesenc_si128(hashA, hashB);
        hashA = _mm_aesenc_si128(hashA, hashC);
        hashA = _mm_aesenc_si128(hashA, hashD);
        length = length - 64;
    }
}
ptr128 = (_mm128i *)buffer;
if (length >= 16) {
    Cycles = length/16;
    for(; Cycles--; buffer += 16) {
        AgainstRules = _mm_loadu_si128(ptr128++);
        GumbotronREVER = _mm_shuffle_epi8 (AgainstRules, ReverseMask);
        GumbotronINTER = _mm_shuffle_epi8 (AgainstRules, InterleaveMask);
        GumbotronREVERINTER = _mm_shuffle_epi8 (GumbotronREVER, InterleaveMask);
        hashA = _mm_aesenc_si128(hashA, AgainstRules);
        hashB = _mm_aesenc_si128(hashB, GumbotronREVER);
        hashC = _mm_aesenc_si128(hashC, GumbotronINTER);
        hashD = _mm_aesenc_si128(hashD, GumbotronREVERINTER);
        hashA = _mm_aesenc_si128(hashA, hashB);
        hashA = _mm_aesenc_si128(hashA, hashC);
        hashA = _mm_aesenc_si128(hashA, hashD);
        length = length - 16;
    }
}
if (length & (16-1)) { // Inhere using Pippip's approach to read past the end ("the dirty" sentinel like style, or more like padding)
    AgainstRules = _mm_loadu_si128(ptr128);
    //AgainstRules = _mm_srli_si128 (AgainstRules, 16-length); // catastrophic error: Intrinsic parameter must be an immediate value
    AgainstRules = _mm_and_si128 (AgainstRules, Gumbotron[16-length]);
    //Gumbotron = _mm_slli_si128 (Gumbotron, 16-length); // catastrophic error: Intrinsic parameter must be an immediate value
    Gumbotron = _mm_and_si128 (hashB, Gumbotron[16-length]);
    AgainstRules = _mm_or_si128 (AgainstRules, Gumbotron);
    GumbotronREVER = _mm_shuffle_epi8 (AgainstRules, ReverseMask);
    GumbotronINTER = _mm_shuffle_epi8 (AgainstRules, InterleaveMask);
    GumbotronREVERINTER = _mm_shuffle_epi8 (GumbotronREVER, InterleaveMask);
    hashA = _mm_aesenc_si128(hashA, AgainstRules);
    hashB = _mm_aesenc_si128(hashB, GumbotronREVER);
    hashC = _mm_aesenc_si128(hashC, GumbotronINTER);
    hashD = _mm_aesenc_si128(hashD, GumbotronREVERINTER);
    hashA = _mm_aesenc_si128(hashA, hashB);
    hashA = _mm_aesenc_si128(hashA, hashC);
    hashA = _mm_aesenc_si128(hashA, hashD);
}
SlowCopy128bit( (const char *)&hashA, (char *)&DDAES[0]);
}

```



Nakamichi's lookupper/keysrinker *DoubleDeuceAES_XMM* - hashing smallkeys (1..64..) down to 16 bytes

// <https://godbolt.org/> icc 19.0.0 -o3 -maxv [

SlowCopy128bit(char const*, char*):

```
vmovdqu xmm0, XMMWORD PTR [rdi]
vmovdqu XMMWORD PTR [rsi], xmm0
ret
```

DoubleDeuceAES_Gumbotron(unsigned char const*, unsigned long):

```
vmovups xmm8, XMMWORD PTR .L2i10floatpacket.0[rip]
vpxor xmm15, xmm15, xmm15
vmovups xmm4, XMMWORD PTR .L2i10floatpacket.1[rip]
vmovups xmm1, XMMWORD PTR .L2i10floatpacket.2[rip]
vmovdqu xmm0, XMMWORD PTR .L2i10floatpacket.3[rip]
vmovdqu xmm7, XMMWORD PTR .L2i10floatpacket.4[rip]
vmovdqu xmm5, XMMWORD PTR .L2i10floatpacket.5[rip]
vmovdqu xmm6, XMMWORD PTR .L2i10floatpacket.6[rip]
vmovdqu xmm3, XMMWORD PTR .L2i10floatpacket.7[rip]
vmovdqu xmm2, XMMWORD PTR .L2i10floatpacket.8[rip]
cmp rsi, 64
jbe .B2.6
mov rax, rsi
shr rax, 6
dec rax
cmp rax, -1
je .B2.7
```

```
..B2.4: vmovdqu xmm7, XMMWORD PTR [48+rdi]
vmovdqu xmm0, XMMWORD PTR [rdi]
vmovdqu xmm5, XMMWORD PTR [16+rdi]
vmovdqu xmm3, XMMWORD PTR [32+rdi]
vmovdqu xmm12, XMMWORD PTR .L2i10floatpacket.3[rip]
dec rax
vpsubf xmm9, xmm7, xmm12
vpsubf xmm13, xmm3, xmm12
vaesenc xmm10, xmm8, xmm0
add rsi, -64
vaesenc add xmm8, xmm10, xmm5
rdi, 64
vaesenc xmm11, xmm4, xmm9
vaesenc xmm4, xmm8, xmm3
vpsubf xmm8, xmm5, xmm12
vpsubf xmm12, xmm0, xmm12
vaesenc xmm14, xmm11, xmm13
vaesenc xmm14, xmm14, xmm8
vaesenc xmm11, xmm4, xmm7
vaesenc xmm10, xmm14, xmm12
XMMWORD PTR [-24+rsp], xmm10
vaesenc xmm14, xmm11, xmm10
vmovdqu xmm10, XMMWORD PTR .L2i10floatpacket.4[rip]
vmovdqu xmm11, XMMWORD PTR .L2i10floatpacket.5[rip]
vpsubf xmm4, xmm0, xmm10
vpsubf xmm2, xmm0, xmm11
vpsubf xmm0, xmm3, xmm6
vpor xmm4, xmm4, xmm0
vaesenc xmm1, xmm1, xmm4
vmovdqu xmm4, XMMWORD PTR .L2i10floatpacket.7[rip]
vpsubf xmm3, xmm3, xmm4
vpsubf xmm0, xmm5, xmm10
vpor xmm2, xmm2, xmm3
vaesenc xmm2, xmm1, xmm2
vpsubf xmm1, xmm5, xmm11
vpsubf xmm5, xmm7, xmm6
vpsubf xmm3, xmm8, xmm6
vpsubf xmm7, xmm7, xmm4
vpsubf xmm8, xmm8, xmm4
vpsubf xmm4, xmm12, xmm4
vpor xmm0, xmm0, xmm5
vpor xmm1, xmm1, xmm7
vaesenc xmm2, xmm2, xmm0
vmovdqu xmm0, xmm9, xmm10
vpsubf xmm10, xmm13, xmm10
vpor xmm5, xmm0, xmm3
vaesenc xmm0, xmm15, xmm5
vpsubf xmm15, xmm9, xmm11
vpsubf xmm11, xmm13, xmm11
vaesenc xmm1, xmm2, xmm1
vpor xmm15, xmm15, xmm8
vpsubf xmm2, xmm12, xmm6
vaesenc xmm3, xmm0, xmm15
vpor xmm5, xmm10, xmm2
vaesenc xmm7, xmm3, xmm5
vpor xmm9, xmm11, xmm4
vaesenc xmm15, xmm7, xmm9
vaesenc xmm13, xmm14, xmm1
vaesenc xmm8, xmm13, xmm15
vmovups xmm4, XMMWORD PTR [-24+rsp]
cmp rax, -1
jne .B2.4
vmovdqu xmm2, XMMWORD PTR .L2i10floatpacket.8[rip]
vmovdqu xmm0, XMMWORD PTR .L2i10floatpacket.3[rip]
```

..B2.6: cmp rsi, 16

jb .B2.11

..B2.7: mov rax, rsi

shr rax, 4

dec rax

cmp rax, -1

je .B2.11

..B2.9: vmovdqu xmm7, XMMWORD PTR [rdi]

vpsubf xmm5, xmm7, xmm0

vpsubf xmm3, xmm7, xmm2

vpsubf xmm6, xmm5, xmm2

vaesenc xmm4, xmm4, xmm5

rax

vaesenc xmm8, xmm8, xmm7

add rdi, 16

vaesenc xmm1, xmm1, xmm3

add rsi, -16

vaesenc xmm9, xmm8, xmm4

vaesenc xmm15, xmm15, xmm6

```
vaesenc xmm10, xmm9, xmm1
vaesenc xmm8, xmm10, xmm15
cmp rax, -1
jne .B2.9
```

```
..B2.11: test rsi, 15
je .B2.13
shl rsi, 4
vmovdqu xmm3, XMMWORD PTR [rdi]
mov rax, QWORD PTR Mumbotron[rip]
mov rdx, QWORD PTR Jumbotron[rip]
vpand xmm5, xmm3, XMMWORD PTR [rsi+rax]
vpand xmm6, xmm4, XMMWORD PTR [rsi+rdx]
vpor xmm7, xmm5, xmm6
vpsubf xmm10, xmm7, xmm0
vpsubf xmm0, xmm7, xmm2
vpsubf xmm2, xmm10, xmm2
vaesenc xmm8, xmm8, xmm7
vaesenc xmm4, xmm4, xmm10
vaesenc xmm9, xmm8, xmm4
vaesenc xmm1, xmm1, xmm0
vaesenc xmm11, xmm9, xmm1
vaesenc xmm12, xmm15, xmm2
vaesenc xmm8, xmm11, xmm12
```

```
..B2.13: vmovups XMMWORD PTR DDAES[rip], xmm8
ret
__sti__: mov QWORD PTR Mumbotron[rip], offset flat: VectorsNeedNonVariable1
mov QWORD PTR Jumbotron[rip], offset flat: VectorsNeedNonVariable2
ret
Mumbotron:
Jumbotron:
DDAES:
VectorsNeedNonVariable1:
...
VectorsNeedNonVariable2:
...
.L2i10floatpacket.0:
.long 0x6295c58d, 0x62b82175, 0x07bb0142, 0x6c62272e
.L2i10floatpacket.1:
.long 0xc4e576cc, 0x2d98c384, 0xaac55036, 0xdd268dbc
.L2i10floatpacket.2:
.long 0xcaee0535, 0x1023b4c8, 0x47b6bbb3, 0x8cb15368
.L2i10floatpacket.3:
.long 0x0c0de0ef, 0x08090a0b, 0x04050607, 0x00010203
.L2i10floatpacket.4:
.long 0x80018000, 0x80038002, 0x80058004, 0x80078006
.L2i10floatpacket.5:
.long 0x80098008, 0x800b800a, 0x800d800c, 0x800f800e
.L2i10floatpacket.6:
.long 0x01800080, 0x03800280, 0x05800480, 0x07800680
.L2i10floatpacket.7:
.long 0x09800880, 0x0b800a80, 0x0d800c80, 0x0f800e80
.L2i10floatpacket.8:
.long 0x09010800, 0x0b030a02, 0x0d050c04, 0x0f070e08
```

// <https://godbolt.org/> icc 19.0.0 -o3 -maxv [

Testfile: TERAPIG_Encyclopaedia_Judaica_(in_22_volumes).TXT.tar (107,784,192 bytes)
Testmachine: Laptop 'Brutalitto' AMD 4800H max turbo 4.3GHz, 64GB DDR4 3200MHz, windows 10
Hashtable: 27bit, i.e. 134,217,728 slots, greater than (107,784,192 bytes), since in case of perfect hasher - slots should be more than the keys at each position

| Hasher, GCC-10.1 compiler -O3 -maxv | Number Of Hash Collisions = Distinct Keys - Number Of Trees | RAW Hashing Speed (in one pass, at each position) for keys 4,6,8,10,12,14,16,18,36,64 bytes |
|---|---|---|
| DoubleDeuceAES_Gumbotron | 135,752,271 | 204,640,573 KEYS-PER-SECOND |
| HighwayHash128 (generic) | 135,754,873 | 6,336,146 KEYS-PER-SECOND |
| XXH3_128bits v0.8.0 | 135,756,978 | 212,843,977 KEYS-PER-SECOND |
| wyhash final | 135,762,454 | 442,100,861 KEYS-PER-SECOND |
| XXH3_64bits v0.8.0 | 135,763,366 | 290,994,033 KEYS-PER-SECOND |
| CRC32C (.mm_crc32_u32) | 135,764,628 | 252,599,460 KEYS-PER-SECOND |
| FNW1A_Pippip | 135,768,302 | 450,602,801 KEYS-PER-SECOND |
| SHA3-224 | 135,771,905 | 153,841 KEYS-PER-SECOND |

Note: The second column houses the cumulative value for all collisions, the collisions for all orders 4..64 were summed, that is.

Excellent speed for a lookupper!
Considering the hash is 128bit, sometimes (in the future) 32bit won't suffice!
The mini-main cycle (for keys 1..63 bytes long), delivering 200+ million key/s hash speed on Zen 2 Renoir 4800H max turbo 4.3GHz, DDR4 3200MHz.

Supernifty speed for a keysrinker!

The main cycle, delivering 8+GB/s linear hash speed on Zen 2 Renoir 4800H max turbo 4.3GHz, DDR4 3200MHz. Notice, the XMM usage has to be changed with ZMM - the ideal etude it would be - one memory access via ZMM, and _mm512_shuffle_epi8(_mm512i a, _mm512i b) in lock step.

DoubleDeuceAES_Gumbotron_XMM

is 31% to 41% faster than XXH128, when hashing 200,000,000 lines/keys, 128 bytes each, in RAM. Download the benchmark at: <https://github.com/Cyan4973/xxHash/issues/568#issuecomment-906113959>