

# LOOK-UP-ER-O-RAMA - In Search for FASTEST hasher for table lookups

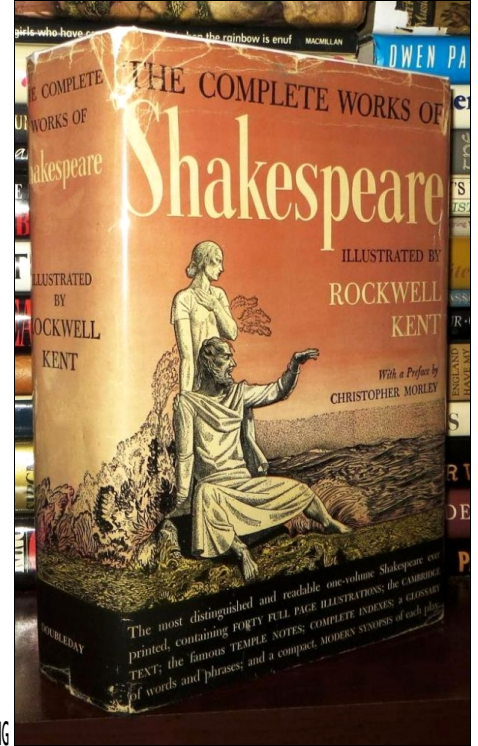
## HASHING Shakespeare (and all English wordlists) ... faster than HARDWARE CRC32

**FNV1A-Deathship-Dualheaded** a.k.a. **FNV1A-Pippip-Yurii** vs **WYHASH** vs **XXH3** showdown, round #3, 2019-Oct-30 by Kaze (at [sanmayce@sanmayce.com](mailto:sanmayce@sanmayce.com))

My word is for FASTEST lookups - the hash functions dedicated to superfast hash-table lookups.  
**FNV1A-Pippip-Yurii** delivers FASTESTNESS, my word.



```
// Dedicated to Pippip, the main character in the 'das Totenschiff' roman, actually the B.Traven himself,
// his real name was Hermann Albert Otto Maksymilian Feige.
// CAUTION: Add 8 more bytes to the buffer being hashed, usually malloc(...+8) - to prevent out of boundary
// reads!
// Many thanks go to Yurii 'Hordi' Hordienko, he lessened with 3 instructions the original 'Pippip', thus:
// #include <stdint.h>
// #define PIPPIP_KAZE(x, n) ((x) << (n)) >> (n)
// #define PIPPIP_YURII(const char* str, size_t wrdlen) {
//     const uint32_t PRIME = 591798841; uint32_t hash32; uint64_t hash64 = 14695981039346656037;
//     size_t Cycles, NHead;
//     if (wrdlen > 8) {
//         Cycles = (wrdlen - 1) >> 4 + 1; NHead = wrdlen - (Cycles < 3);
//         #pragma nounroll
//         for (; Cycles--; str += 8) {
//             hash64 = (hash64 ^ (*(uint64_t*) (str))) * PRIME;
//             hash64 = (hash64 ^ (*(uint64_t*) (str+NHead))) * PRIME;
//         }
//     } else {
//         hash64 = (hash64 ^ PIPPIP_KAZE(*(uint64_t*) (str+0), (8-wrdlen)<3)) * PRIME;
//         hash32 = (uint32_t) (hash64 ^ (hash64>>32)); return hash32 ^ (hash32 >> 16);
//     } // Last update: 2019-Oct-30, 14 C lines strong, Kaze.
```



Credit: The digital artist [Sergey Samuilov](#)

<https://www.codeproject.com/Articles/716530/Fastest-Hash-Function-for-Table-Lookups-in-C>  
<https://software.intel.com/en-us/forums/intel-moderncode-for-parallel-architectures/topic/824947#comment-1947234>

A hidden beauty with deadly appearance, hundreds of billions of English n-grams await to be hashed in SUPERFAST and SUPERDISPERSY way, ENFUN!

**FNV1A-Deathship-Dualheaded** a.k.a. **FNV1A-Pippip-Yurii** a.k.a. **FNV1A-Totenköpfe** - SCREAMING PERFORMANCE



Credit: The stunning art from the Ukrainian tattoo master [Dmitry Yavtushenko](#), Kaze dualheaded it.

Hashing (on i7-3630QM) the whole book (5,784,758 bytes) in chunks a.k.a. Building-Blocks, (Slots=1<Bits): 23, meaning 10x2x23 total slots in 10 hashtables or 10 x 8,388,608 slots.  
Note1: RAW (i.e. 'ON THE FLY') Hashing Speed is given - one order at a time, without touching any SLOTS (uncached RAM).  
Note2: Last column contains CUMULATIVE Collisions i.e. TOTAL ones.

Hasher / Speed in Keys/s for	4 bytes	6 bytes	8 bytes	10 bytes	12 bytes	14 bytes	16 bytes	18 bytes	36 bytes	64 bytes	Collisions
FNV1A-Pippip-Yurii (Intel v19.0)	262,943,409	262,943,318	262,943,227	289,237,450	289,237,350	289,237,250	289,237,150	222,490,038	180,772,593	144,617,075	9,090,483
XXH3 (Intel v19.0)	214,250,185	206,598,321	214,250,037	206,598,178	206,598,107	199,473,965	199,473,896	160,687,250	111,244,673	111,244,134	9,097,678
WYHASH (Intel v19.0)	170,139,852	165,278,657	160,687,527	160,687,472	160,687,416	148,326,794	156,344,405	131,471,386	103,298,625	73,223,987	9,092,905

The hash scanner - hashing in one pass all orders - TESTING CUMULATIVE PERFORMANCE, without touching any SLOTS (uncached RAM):

**FNV1A-Pippip-Yurii** (Intel v19.0) : RAW Hashing Speed for keys 4,6,8,10,12,14,16,18,36,64 bytes long = 264,144,246 KEYS-PER-SECOND; all in 219 clocks  
**XXH3** (Intel v19.0) : RAW Hashing Speed for keys 4,6,8,10,12,14,16,18,36,64 bytes long = 186,605,129 KEYS-PER-SECOND; all in 310 clocks  
**WYHASH** (Intel v19.0) : RAW Hashing Speed for keys 4,6,8,10,12,14,16,18,36,64 bytes long = 139,391,783 KEYS-PER-SECOND; all in 415 clocks

Having played a lot with **FW** variants, wanna say thanks to:  
- Landon 'chongo' Curt No11, [https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Jvo\\_hash\\_function](https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Jvo_hash_function)  
- Peter Kankowski, [https://www.strchr.com/hash\\_functions](https://www.strchr.com/hash_functions)  
Thank you!

Announcements: <https://twitter.com/Sanmayce/status/1181013776377749505> Look-up-er-rama: <https://github.com/wanqifudan/wyhash/issues/29#issuecomment-540811661>

Using superprecise Peter's benchmark for  
'The Complete Works of William Shakespeare  
by William Shakespeare' 5.51 MB strong,  
line-by-line, Time Collisions:

**KAZE\_www.gutenberg.org\_ebooks\_100.txt:**

138578 lines read	
524288 elements in the table (19 bits)	
Jesteress:	24134 [31211]
Meiyan:	24653 [31116]
Pippip_Yurii:	17178 [31196]
Pippip:	18174 [31196]
Totenschiff:	20336 [31134]
Yorikke:	23733 [31139]
Yoshimura:	19560 [31245]
wyhash:	28118 [31260]
YoshimitsuTRIAD:	27006 [31316]
FNV-1a:	49246 [31178]
Larson:	48988 [31406]
CRC-32:	41789 [31210]
Murmur3:	31513 [31308]
XXHstrong32:	27180 [31118]
iSCSI CRC:	22606 [31248]

The executable (compiled with Intel V19.0 as 64bit) was run on laptop with i5-7200U @3.1GHz turbo.

For good measure - all Wikipedia words:  
**enwiki-20190920-pages-articles.xml.wrd:**  
42206534 lines read

134217728 elements in the table (27 bits)	
Pippip_Yurii:	8253384 [5996107]
Yorikke:	9271283 [6011605]
wyhash:	11241704 [5991525]
CRC-32:	11570725 [5843653]
Murmur3:	11315227 [5992379]
XXHstrong32:	12389203 [6007552]
iSCSI CRC:	8433784 [5803092]

<https://godbolt.org/z/i40ipj> x86-64 gcc 9.2 -O3

**FNV1A-Pippip-Yurii:**

```
mov    rax, QWORD PTR [rdi]
cmp    rsi, 8
jbe    .L2
lea    rax, [rsi-1]
shr    rax, 4
lea    rdx, [8+rax*8]
movabs rax, -3750763034362895579
sub    rsi, rdx
add    rdx, rdi

.L3:
xor     rax, QWORD PTR [rdi]
add     rdi, 8
imul    rax, rax, 591798841
xor     rax, QWORD PTR [rdi+8+rsi]
imul    rax, rax, 591798841
cmp     rdi, rdx
jne     .L3

.L4:
mov     rdx, rax
shr     rdx, 32
xor     eax, edx
mov     edx, eax
shr     edx, 16
xor     eax, edx
ret

.L2:
movabs  rdx, -3750763034362895579
mov     ecx, 8
sub     ecx, esi
sal     ecx, 3
sal     rax, cl
shr     rax, cl
xor     rax, rdx
imul    rax, rax, 591798841
jmp     .L4
```

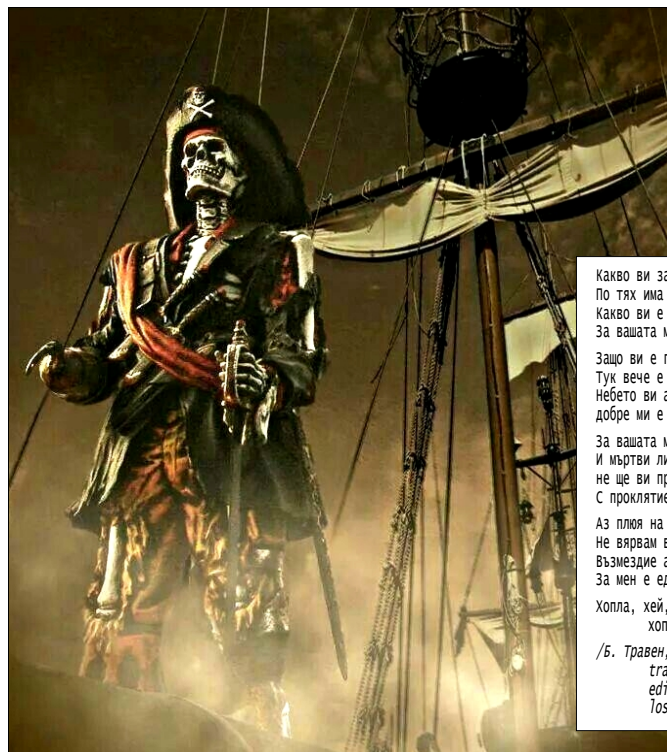
全壊 - ZENKAI -

Bringing death in all departments, **FNV1A-Pippip-Yurii** outspeeds **XXH3** and **WYHASH** in ... Shakespeariada.



# HASHING Shakespeare ... faster than HARDWARE CRC32

FNW1A-Deathship vs WYHASH showdown, 2019-Oct-14 by Kaze



Какво ви засягат моите дрипи?  
По тях има радост и много сълизи.  
Какво ви е грижа дали аз ридая?  
За вашата милост аз не пълзя.

Защо ви е грижа какво ми харесва?  
Тук вече е моят, не вашият свят.  
Небето ви аз не искам да зная,  
добре ми е само в горящия ад.

За вашата милост не искам да зная.  
И мъртви ли нося или богове,  
не ще ви призная. Какво ви засяга?  
С проклятие никой не ще ви зове.

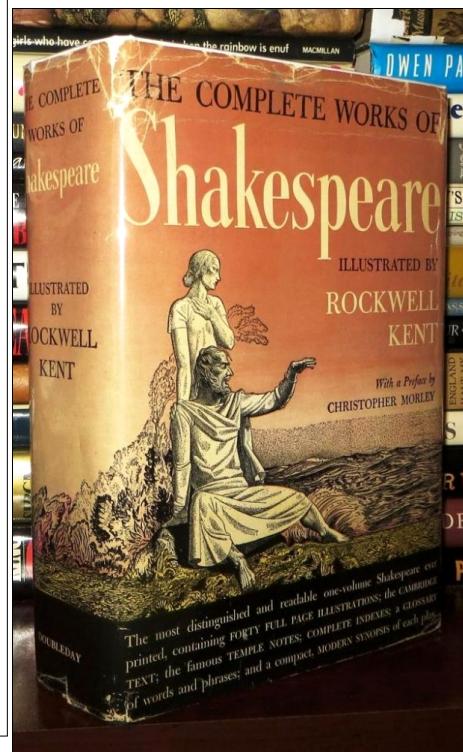
Аз пляя на мита за бога възкръснал.  
Не вярвам в небето и Страшния съд.  
Възмездие адско не ще ме уплаши.  
За мен е единствен широкият път,

Хопла, хей, в открито море, хопла,  
хопла, хей!

/Б. Травен, 'Корабът на Мъртвите',  
translated from the German  
edition, these verses are  
lost in the English one/

```
#define _PAD_KAZE(x, n) ((x) << (n)) >> (n)
uint32_t FNW1A_Hash_TotenschiFF_v1(const char *str, uint32_t hash64) {
    const uint32_t PRIME = 591798841; uint32_t hash32; uint64_t hash64 = 1469598103934656037; const char *p = str;
    for (; width > 2 * sizeof(uint32_t); width -= 2 * sizeof(uint32_t), p += 2 * sizeof(uint32_t))
        hash64 = (hash64 * ((uint64_t *) (p+0)) * PRIME;
        hash64 = (hash64 * ((uint64_t *) (p+4)) * PRIME;
        hash32 = (uint32_t) (hash64 >> 32); return hash32 ^ (hash32 >> 16);
} // Last update: 2019-Oct-14, 8 C lines strong, kaze.
```

hashing by words, Building-Blocks and line-by-line



My word is for FASTEST lookups - the hash functions dedicated to superfast hash-table lookups.

FNW1A-TotenschiFF delivers FASTESTNESS, my word.

Is there a faster one?! In order to find out, I again compared it to the fastest (on the Reini Urban's roster) WYHASH...

<https://github.com/rurban/smhasher>

Once, while reading I have been hit by an ultrab Haiku, it goes like this:

winter  
The shoemaker's boy  
Drinks tea  
It is cold

Instantly I pictured a Zen master walking by the shoemaker's workshop and "compressing" the feeling of coldness into 3 verses.

Such simplicity had always captivating effect on me, Zen stuff. Similar poetry in action as this one, written in another language - Assembler:

```
cmp esi, 8
jbe .L4
lea ecx, [rsi-9]
shr ecx, 3
mov eax, ecx
lea rdx, [rdi+8+rax*8]
movabs rax, -3750763034362895579
.L3: xor rax, QWORD PTR [rdi]
add rdi, 8
imul rax, rax, 591798841
cmp rdi, rdx
jne .L3
sal ecx, 3
neg ecx
lea esi, [rsi-8+rcx]
.L2: mov ecx, 8
mov rdx, QWORD PTR [rdx]
sub ecx, esi
sal ecx, 3
sal rdx, cl
shr rdx, cl
xor rax, rdx
imul rax, rax, 591798841
mov rdx, rax
shr rdx, 32
xor eax, edx
mov edx, eax
shr edx, 16
xor eax, edx
ret
.L4: movabs rax, -3750763034362895579
mov rdx, rdi
jmp .L2
```

No mumbo-jumbo, only QWORD  
fetching, as it should!  
It will get faster with  
incoming  
architectures...

Sanmayce's Haikuoids inspired by "the first kill" scene from "The Hurt Locker" -  
this movie captured so much Zen in battlefield:

Gunlight } Night  
Daylight } Skylight  
Sunset } Key Sea  
Loss } Deathship sailing

Source: [https://forum.thefreedictionary.com/postst29340p2\\_The-latest-in-words-.aspx](https://forum.thefreedictionary.com/postst29340p2_The-latest-in-words-.aspx)

## TRIJECTA

- Fastest words hashing;
- Fastest verses hashing;
- Fastest small chunks hashing.

FNW1A-TotenschiFF was hashing VERSES  
(25689-19977)/19977\*100= 28%  
faster than WYHASH, these  
were smallkeys, not tinykeys.

Above "verses" compress lossfully all types of smallkeys (chunks of data up to some 32 bytes) SUPERFAST - in order to retrieve them by looking up the hashtable(s) later on.  
The second most important sought after property is dispersion/distribution - to avoid stacking collisions in certain slots, "Deathship" seems being on par with strong hashers,  
therefore above verses are both superfast and ... superdispersy :P

Having played a lot with FNW variants, wanna say thanks to:

- Landon 'chongo' Curt Noll, [https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Jevons\\_hash\\_function](https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Jevons_hash_function)

- Peter Kankowski, [https://www.strchr.com/hash\\_functions](https://www.strchr.com/hash_functions)

Thank you!

Hashing (on i7-3630QM) the whole book in chunks a.k.a. Building-Blocks, (Slots=1<<Bits): 24, meaning 10x24 total slots in 10 hashtables or 10 x 16,777,216:

Note1: The underlined values are the better ones.

Note2: Last column contains CUMULATIVE Collisions i.e. TOTAL ones.

Hasher / Speed in Keys/s for	4 bytes	6 bytes	8 bytes	10 bytes	12 bytes	14 bytes	16 bytes	18 bytes	36 bytes	64 bytes
FNW1A-TotenschiFF (Intel v19.0)	9,838,018	6,470,640	6,180,289	5,331,565	5,356,247	5,096,691	5,452,161	5,436,786	5,137,409	4,935,746
WYHASH (Intel v19.0)	8,444,897	6,470,640	5,813,820	5,155,747	5,155,745	5,074,337	5,366,180	5,273,237	5,004,085	4,733,792
FNW1A-TotenschiFF (GCC v7.3.0)	10,460,678	6,377,897	5,896,790	4,812,603	4,323,428	4,852,973	5,278,050	4,605,685	4,182,735	3,986,695
WYHASH (GCC v7.3.0)	10,460,678	6,349,893	5,733,152	5,123,781	4,382,384	4,117,256	5,078,791	4,460,093	4,059,454	3,808,225

Announcements: <https://twitter.com/Sanmayce/status/1181013776377749505> Look-up-er-o-rama: <https://github.com/wangqi-fudan/wyhash/issues/29#issuecomment-540811661>

Page 2 of 17



# Nakamichi's lookuperess *FNV1A-Totenschiff* - hashing (in plain C) smallkeys faster than SSE4.2 iSCSI-CRC32



FNV1A-Totenschiff a.k.a. 'Парцалышка' a.k.a. 'Partszalyschka'  
FNV1A-Yorikke a.k.a. 'Прахоляшка' a.k.a. 'Praxolyaschka'



```
// "There it now stands for ever. Black on white. I can't get away from it. Ahoy, Yorikke, ahoy, hoy, ho!  
// Go to hell now if you wish. What do I care? It's all the same now to me.  
// I am part of you now. Where you go I go, where you leave I leave, when you go to the devil I go. Married.  
// Vanished from the living. Damned and doomed. Of me there is not left a breath in all the vast world.  
// Ahoy, Yorikke! Ahoy, hoy, ho! / I am not buried in the sea, / The death ship is now part of me / So far from sunny New Orleans / So far from lovely Louisiana."  
// /An excerpt from 'THE DEATH SHIP - THE STORY OF AN AMERICAN SAILOR' by B. TRAVEN/  
// "walking home to our good old Yorikke, I could not help thinking of this beautiful ship, with a crew on board that had faces as if they were seeing ghosts by day and  
// Compared to that gilded Empress, the Yorikke was an honorable old lady with lavender sachets in her drawers.  
// Yorikke did not pretend to anything she was not. She lived up to her looks. Honest to her lowest ribs and to the leaks in her bilge.  
// Now, what is this? I find myself falling in love with that old jane.  
// All right, I cannot pass by you, Yorikke; I have to tell you I love you. Honest, baby, I love you.  
// I have six black finger-nails, and four black and green-blue nails on my toes, which you, honey, gave me when necking  
// Grate-bars have crushed some of my toes. And each finger-nail has its own painful story to tell.  
// My chest, my back, my arms, my legs are covered with scars of burns and scorchings.  
// Each scar, when it was being created, caused me pains which I shall surely never forget.  
// But every outcry of pain was a love-cry for you, honey.  
// You are no hypocrite. Your heart does not bleed tears when you do not feel heart-aches deeply and truly.  
// You do not dance on the water if you do not feel like being jolly and kicking chasers in the pants.  
// Your heart never lies. It is fine and clean like polished gold. Never mind the rags, honey dear.  
// When you laugh, your whole soul and all your body is laughing.  
// And when you weep, sweetly, then you weep so that even the reefs you pass feel like weeping with you.  
// I never want to leave you again, honey. I mean it. Not for all the rich and elegant buckets in the world.  
// I love you, my gypsy of the sea!"  
// /An excerpt from 'THE DEATH SHIP - THE STORY OF AN AMERICAN SAILOR' by B. TRAVEN/  
#define _rotl_KAZE(x, n) (((x) << (n)) | ((x) >> (32-(n))))  
#define _PADR_KAZE(x, n) ((x) << (n)) >> (n)  
#define ROLInBits 27 // 5 in r.1; Caramba: it should be ROR by 5 not ROL, from the very beginning the idea was to mix two  
// CAUTION: Add 8 more bytes to the buffer being hashed, usually malloc(...+8) - to prevent out of boundary reads!  
uint32_t FNV1A_Hash_Yorikke_v3(const char *str, uint32_t wrdlen) {  
    const uint32_t PRIME = 591798841; uint32_t hash32 = 2166136261; uint64_t PADDEDby8; const char *p = str;  
    for(; wrdlen > 2*sizeof(uint32_t); wrdlen -= 2*sizeof(uint32_t), p += 2*sizeof(uint32_t)) {  
        hash32 = (_rotl_KAZE(hash32, ROLInBits) ^ *(uint32_t *) (p+0)) * PRIME;  
        hash32 = (_rotl_KAZE(hash32, ROLInBits) ^ *(uint32_t *) (p+4)) * PRIME;  
    }  
    // Here 'wrdlen' is 1..8  
    PADDEDby8 = _PADR_KAZE(*(uint64_t *) (p+0), (8-wrdlen)<<3); // when (8-8) the QWORD remains intact  
    hash32 = (_rotl_KAZE(hash32, ROLInBits) ^ *(uint32_t *) ((char *) &PADDEDby8+0)) * PRIME;  
    hash32 = (_rotl_KAZE(hash32, ROLInBits) ^ *(uint32_t *) ((char *) &PADDEDby8+4)) * PRIME;  
    return hash32 ^ (hash32 >> 16);  
}  
#define _PADR_KAZE(x, n) ((x) << (n)) >> (n)  
uint32_t FNV1A_Hash_Totenschiff_v1(const char *str, uint32_t wrdlen) {  
    const uint32_t PRIME = 591798841; uint32_t hash32; uint64_t hash64 = 14695981039346656037; const char *p = str;  
    for(; wrdlen > 2*sizeof(uint32_t); wrdlen -= 2*sizeof(uint32_t), p += 2*sizeof(uint32_t)) {  
        hash64 = (hash64 ^ *(uint64_t *) (p+0)) * PRIME;  
        hash64 = (hash64 ^ (_PADR_KAZE(*(uint64_t *) (p+0), (8-wrdlen)<<3))) * PRIME;  
        hash32 = (uint32_t)(hash64 ^ (hash64 >> 32)); return hash32 ^ (hash32 >> 16);  
    } // Last update: 2019-Oct-14, 8 C lines strong, kaze.
```

<https://github.com/wangyi-fudan/wyhash/issues/29#issuecomment-538078396>  
<https://software.intel.com/en-us/forums/intel-moderncode-for-parallel-architectures/topic/824947#comment-1946227>  
<https://twitter.com/Sanmayce/status/1179837962135261184>  
[https://www.strchr.com/hash\\_functions#comment-777](https://www.strchr.com/hash_functions#comment-777)  
<https://cblomrants.blogspot.com/2010/11/11-29-10-useless-hash-test.html?showComment=1570129121322#c1973661950653692920>  
<https://www.overclock.net/forum/21-benchmarking-software-discussion/1319572-benchmarking-fastest-hash-function.html#post28146508>

This PC

Shortcut

64

AIDA64

Extreme

calibre 64bit

E-book man...

cpu-z v64.exe

- Shortcut

D:\PeterK\_strchr.com\ISCSI-CRC\_vs\_WYHASH\_vs\_FNV1A-Yorikke-v3\hash\_ICL\_v15\_64bit.exe KAZE\_www.byronknoll.com\cmix-v18.zip\_english.dic

44880 lines read

131072 elements in the table (17 bits)

Jesteress:511943684588448746984519448545094480463243686721

Meiyant:509145564557477448074622453245615408460545326923

Yorikke:320031863345336931793162313831493406364631386883

Yoshimura:428342534229449242944311442742704442429742297013

Yoshimitsu:492245775863511455515240471448374765478345776812

YoshimitsuTRIAD:446145314466509543804402449044234348435643487006

FNV-1a:445549494944444944184517453744414398440443986833

Larson:552146724496452444884743456044914608562244886830

CRC-32:478645554504448344904715452748094969456144836891

Murmur2:485647724744472749264959523650404812487847276820

Murmur3:451745864519461857044609448545414591452844856874

XXHfast32:496650055472531249514959491750565002496949176812

XXHstrong32:885364095056501050035115512267785100593650036819

ISCSI CRC:374337023691370437093835432638843701368236826785

D:\PeterK\_strchr.com\ISCSI-CRC\_vs\_WYHASH\_vs\_FNV1A-Yorikke-v3\hash\_ICL\_v15\_32bit.exe KAZE\_www.byronknoll.com\cmix-v18.zip\_english.dic

44880 lines read

131072 elements in the table (17 bits)

Jesteress:544651325036502950805048503651465026517650266721

Meiyant:512153705109514051636607603260056039597351096923

Yorikke:4501460117946453348334487449845345304499444876883

Yoshimura:502849284964511850155056598960055965585949287013

Yoshimitsu:14161135081171811519117801061810416104121135710573104126812

YoshimitsuTRIAD:467946924718468546664915478946874667466246627006

FNV-1a:486649094875485854304856487148604987486248566833

Larson:483148493363485448054816482148464872481248056830

CRC-32:539949505400552856145542551754906412554349506891

Murmur2:603260026502603959676721611160075994600159676820

Murmur3:62365738734367396756691667967176516565256526874

XXHfast32:626068616284623065736455628363716265631062656812

XXHstrong32:615170696944611961116102616960936818645660936819

ISCSI CRC:482947724738475648365104558853705388478047386785

<http://gcc.godbolt.org/x86-64 gcc 8.3, -O3>

```
FNV1A_Hash_Yorikke_v3:  
cmp esi, 8  
jbe .L4  
lea ecx, [rsi-9]  
shr ecx, 3  
mov eax, ecx  
lea rdx, [rdi+rax*8]  
mov eax, -2128831035  
.L3:  
ror eax, 5  
xor eax, DWORD PTR [rdi]  
add rdi, 8  
imul eax, eax, 591798841  
ror eax, 5  
xor eax, DWORD PTR [rdi-4]  
imul eax, eax, 591798841  
cmp rdi, rdx  
jne .L3  
neg ecx  
ror eax, 5  
lea esi, [rsi-8rcx*8]  
.L2:  
mov ecx, 8  
mov rdx, QWORD PTR [rdx]  
sub ecx, esi  
sal ecx, 3  
sal rdx, cl  
shr rdx, cl  
xor eax, edx  
shr rdx, 32  
imul eax, eax, 591798841  
ror eax, 5  
xor eax, edx  
imul eax, eax, 591798841  
mov edx, eax  
shr edx, 16  
xor eax, edx  
ret  
.L4:  
mov rdx, rdi  
mov eax, 738780398  
jmp .L2
```

<http://gcc.godbolt.org/x86-64 gcc 8.3, -O3>

```
FNV1A_Hash_Yorikke_v3:  
mov ecx, esi  
mov r9, rdi  
mov r8d, ecx  
edx, -2128831035  
cmp ecx, 8  
jbe .B1.8  
mov eax, 1  
lea esi, DWORD PTR [-1+rcx]  
xor esi, edi  
shr esi, 4  
je .B1.4  
.B1.4:  
shld edx, edx, 27  
xor edx, DWORD PTR [r9]  
inc edi  
imul eax, edx, 591798841  
shld eax, eax, 27  
xor eax, DWORD PTR [4+r9]  
imul edx, eax, 591798841  
shld edx, edx, 27  
xor edx, DWORD PTR [8+r9]  
r10d, edx, 591798841  
r10d, r10d, 27  
xor r10d, DWORD PTR [12+r9]  
r9, 16  
imul r10d, r10d, 591798841  
cmp edi, esi  
jbe .B1.4  
mov eax, edi  
shl eax, 4  
neg eax  
add ecx, eax  
lea eax, DWORD PTR [1+rdi+rdi]  
.B1.6:  
lea edi, DWORD PTR [-1+r8]  
shr edi, 3  
lea esi, QWORD PTR [-1+rax]  
cmp esi, edi  
jae .B1.8  
shld edx, edx, 27  
xor edx, DWORD PTR [r9]  
imul edx, edx, 591798841  
shl eax, 3  
neg eax  
shld edx, edx, 27  
xor edx, DWORD PTR [4+r9]  
r9, 8  
imul edx, edx, 591798841  
lea ecx, DWORD PTR [r8+rax]  
.B1.8:  
neg ecx  
shl ecx, 3  
mov rax, QWORD PTR [r9]  
shr rax, cl  
shld edx, edx, 27  
shr rax, cl  
xor eax, edx  
imul edx, edx, 591798841  
shld edx, edx, 27  
shr rax, 32  
xor eax, edx  
imul eax, edx, 591798841  
mov esi, eax  
shr esi, 16  
xor eax, esi  
ret
```

Simply, **Yorikke** and **Totenschiff** are the fastest 32bit hashers FOR LOOKUP-TABLES in 32bit code and 64bit code, respectively, ENFUN!  
'Прахоляшка' and 'Парцалышка' dust 'em all off.