PERFECT PANGRAM HASH : Anagram Hash Function

Byrl Raze Buckbriar

octade.net | sci.crypt | archive.org/details/@buckbriar

1 February 2026

A pangram is a sentence or phrase that contains each letter of an alphabet or character set at least once. A perfect pangram is an anagram of the alphabet which contains each letter exactly once.

Pangram hash generates a perfect pangram hash digest consisting of a anagram permutation of a character set. Each character in the output is unique and non-repeating.

Below is purely procedural source code written in the FreePascal programming language. Instructions and explanatory comments are located in the source code. A plain text source file is embedded in the PDF document. Additional resource files are also embedded in the PDF file.

- -

```
program pangram58_hash(input, output);

{ +++ Copyright (c) 2026, Raze (OCTADE). +++ https://octade.net +++ }

{ Copious comments in the code describe the operations below. }

{ %%% Test vectors are located at the end of this source file. %%% }

{ Output: 58-character permutation set. }
{ Arch: Unix-like, 64-bit. }
{ Compiling : 'fpc -O4 [filename]'. }
{ Installing : Copy to $PATH directory. }
{ Usage : Cat or pipe a file to the program. }
{ Usage : pangram [filename(s)] }

{$INFO >    (PG58) PANGRAM HASH FUNCTION DEMO   +-----------------+   }
{$INFO >                                        |     OCTADE      |   }
{$INFO >     This software is only a demo.      +-----------------+   }
{$INFO >     It is for research use only.       |    sci.crypt    |   }
{$INFO >     No warranty. No guarantee.         +-----------------+   }
{$INFO >     Use entirely at your own risk.     |   octade.net    |   }
{$INFO >                                        +-----------------+   }

{
  Shuffle a pangram set of unique byte characters to produce a digest.

  The hash digest always contains the same character set in random order.

  Inputs relocate values rather than changing them.

  This hash is meant for generating unique identifiers from document metadatas.

  This is a unique experimental construction that I haven't seen yet elsewhere.

  One goal is to process input as single bytes rather than big words.

  Another goal is to have reasonable speed with the single-byte restriction.
}

{$MODE OBJFPC}
```

```
USES
    SysUtils;

TYPE
    B = BYTE; { Cleaner code block. }
    Q = QWORD;

CONST
    { Include desired unique characters in this set. More chars = more bits. }
    { Bits: Hex = 44. Base32 = 117. Base36 = 138. Base58 = 260. Base62 = 284. }
    { Customize charset via the pangram constant in the digest procedure. }

    pangram58 : set of char =
    ['1','2','3','4','5','6','7','8','9','a','A','b','B','c','C',
     'd','D','e','E','f','F','g','G','h','H','i','j','J','k','K',
     'L','m','M','n','N','o','p','P','q','Q','r','R','s','S','t',
     'T','u','U','v','V','w','W','x','X','y','Y','z','Z'];

    { == N.U.M.S. == (N)othing (U)p (M)y (S)leeve Magic Numbers == }
    { Extra magic numbers /could/ be user-specified HMAC or secret key values. }
    MAGIC : Q = Q ($5C00B1ED00B1ED00 + $C0CAC01AC1A551C + $BA5EBA11F007BA11) ;

VAR
    pc : Cardinal; { Size of file list params. }

    FileName: ANSISTRING;
    CBUF: array [0..255] of CHAR;  { Input block chars. }
    FUB_ : array [0..255] of BYTE absolute CBUF; { Input block bytes. }
    NumCharsRead: Integer; { Input byte counter. }
    tot: QWORD ; { Input byte total counter. }
    SkipFile : BOOLEAN;

    CharFile : file of char;
    i : word; { Common looper. }
    x : byte ; { reduce operations in loop. }
    s1, s2 : BYTE; { Input and swapper bytes. }
    ch : char; { Digest output characters. }
    mix : array [0..255] of BYTE; { Mixing state array and entropy pool. }
    e : array [0..255] of BYTE; { Chaotic swap instructions and reflector. }

PROCEDURE SWAP_HASH_BYTES; { This is the simple core hashing function. }
begin; { Turn off compiler range checks if this causes an error. }

x := B(i * 8) ; { Multiply for less loops with addition for indexes. }

inc (e[B(x+0)], FUB_[B(x+0)] + mix[e[B(x+7)]]); { Add input and map byte  }
inc (e[B(x+1)], FUB_[B(x+1)] + mix[e[B(x+6)]]); { to the chaos array.     }
inc (e[B(x+2)], FUB_[B(x+2)] + mix[e[B(x+5)]]); { Recurse into the chaos  }
inc (e[B(x+3)], FUB_[B(x+3)] + mix[e[B(x+4)]]); { array for better mixing }
inc (e[B(x+4)], FUB_[B(x+4)] + mix[e[B(x+3)]]); { and to increase the     }
inc (e[B(x+5)], FUB_[B(x+5)] + mix[e[B(x+2)]]); { difficulty of inverting }
inc (e[B(x+6)], FUB_[B(x+6)] + mix[e[B(x+1)]]); { and pre-imaging.        }
inc (e[B(x+7)], FUB_[B(x+7)] + mix[e[B(x+0)]]);

{ Swap 8 bytes using the same cross-array traversal method as above. }
s1 := mix[e[B(x+0)]] ; s2 := mix[e[B(x+1)]] ; mix[e[B(x+0)]] := s2 ; mix[e[B(x+1)]] := s1 ;
s1 := mix[e[B(x+2)]] ; s2 := mix[e[B(x+3)]] ; mix[e[B(x+2)]] := s2 ; mix[e[B(x+3)]] := s1 ;
s1 := mix[e[B(x+4)]] ; s2 := mix[e[B(x+5)]] ; mix[e[B(x+4)]] := s2 ; mix[e[B(x+5)]] := s1 ;
s1 := mix[e[B(x+6)]] ; s2 := mix[e[B(x+7)]] ; mix[e[B(x+6)]] := s2 ; mix[e[B(x+7)]] := s1 ;

end; { //////// PROCEDURE SWAP_HASH_BYTES //////// }

PROCEDURE EXPORT_DIGEST;
begin;

fillchar(FUB_[0], Length(FUB_), #0); { Zero input block to prevent replay. }

for i := 0 to 255 do
begin;
e[i] += byte(RorQword(tot, i)) ; { Use the counter to account for zeros. }
end;
```

```
{ Mix it up for the digest. 256 * 8 = 2048 chaos sums and byte swaps. }
for i := 0 to 255 do begin; SWAP_HASH_BYTES; end;

for i := 0 to 255 do
begin;
ch := char(mix[i]);
if ch in pangram58 then write(ch); { Change pangram array to preferred charset. }
end;

write('  ');
if FileName = '' then FileName := '-';
writeln (FileName);

end; { //////// PROCEDURE EXPORT_DIGEST //////// }

PROCEDURE READ_LOOP; { TODO : check hash digest file entries. }
begin;

if DirectoryExists(FileName) then begin;
writeln('pangram58 : is a directory : ', FileName) ; SkipFile := True; exit ; end;

FileMode := 0;

try;
ASSIGN(CharFile, FileName);
RESET(CharFile, SizeOf(char));
except;
writeln('pangram58 : error : glob or file not found: ', #39, FileName, #39);
SkipFile := True;
end;

NumCharsRead := 256;

if SkipFile = True then exit;

{$I-}
while (NumCharsRead = 256) and (SkipFile = False) do
begin;
fillchar(FUB_[0], Length(FUB_), #0); { Zero input block to prevent replay. }
try;
BlockRead(CharFile, CBUF, Length(CBUF), NumCharsRead);
except;
writeln('pangram58 : error : file read error 1');
exit;
end;
inc(tot, NumCharsRead);

    { Hash the input data array into the pool. }
    { For more strength increase loop to 63, 127, or 255, re-using each input. }
    { For more speed increase input array size and increase loop span. }
    for i := 0 to 31 do
    begin;
        SWAP_HASH_BYTES;
    end; { for i }
end;
{$I+}
CLOSE(CharFile);

end; { //////// PROCEDURE READ_LOOP //////// }

PROCEDURE INITIALIZE;
begin;

SkipFile := False;

(* Initialize zero values and magic N.U.M.S. *)

fillchar(mix, SizeOf(mix), #0);
fillchar(e, SizeOf(e), #0);
fillchar(CBUF, SizeOf(CBUF), #0);
fillchar(FUB_, SizeOf(FUB_), #0);
```

```
tot := 0; s1 := 0; s2 := 0; NumCharsRead := 0 ; i := 0 ;

{ == N.U.M.S. == (N)othing (U)p (M)y (S)leeve Magic Numbers == }
{ TODO : move to array copy of CONST header }

{ Generate entropy for arrays of unique byte per index. }
{ These could be pre-computed in CONST/VAR headers for slight speed gain. }
for i := 0 to 255 do begin; mix[i] := byte(MAGIC*i); end;
for i := 0 to 255 do begin; e[i] := byte(MAGIC*i*13); end;
for i := 0 to 255 do begin; FUB_[i] := byte(MAGIC*i*29); end;

{ Mix the array to a slightly more pseudorandom state. }
{ Mix it up for the digest. 1024 * 8 = 8192 chaos sums and byte swaps. }
for i := 0 to 1023 do begin; SWAP_HASH_BYTES; end;

{ We now have an initial pangram array of pseudorandomly mixed, unique bytes. }
{ Each byte value from 0 to 255 occurs exactly and only once in the array. }
{ The hashing function will pseudorandomly re-index and anagram the array. }
{ The digest function exports only the specified charset in index order. }

end; { //////// PROCEDURE INITIALIZE //////// }

BEGIN { MAIN ROUTINE }

if ParamCount = 0 then { Check for read from standard input. }
begin;
FileName := '';
INITIALIZE;
READ_LOOP;
if SkipFile = False then EXPORT_DIGEST;
end;

if ParamCount > 0 then { Check for filenames in the command. }
begin;
for pc := 1 to ParamCount do
begin;
FileName := ParamStr(pc);
INITIALIZE;
READ_LOOP;
if SkipFile = False then EXPORT_DIGEST;
end;
end;

END. { MAIN ROUTINE }

(*   //////////   TEST VECTORS   ////////

[null input]    iTv4cgwrEUq5dGJjeWZPMtFH72fyhYBbVNCRAzaK89msunD6LkxQSop31X
[zero byte]     62g1YcBNXrLnUvjhyGTPwW8V7C9dFoK43Rxte5bzEkQfuaZspiMDmSqAJH
[2 zero bytes]  p37RgP6JWLYtQu2oydr84aqFnsTzABx5MXZEKeS19hjkbvVwHDUCfNimGc
['hello world'] wZXDF3WfyqjG47rJEo5HkMRYUSmNgub8e2LspKhCvaP9VitdTBzAnx1cQ6
['pangram']     Zeyp7QNkjbV4FWRdLmYcDiXns2vwEtCaxKT6hB5g1M8fuJUorSAGq9HP3z
['PANGRAM']     K5QaEsjzpf9bJqcoN2V1dvxC3hryYMnDS6e8WBwk7HuPRZiFmLtUXAgGT4
['hash']        AM1CzNT3ED4SirFbPvRJUcexpwtBjguV9oy2kYhQGfZWX76dm5n8HaKLsq
['0']           Cthe2Z5vHzigwaoqn731pAs9WEQM6JruSyPXdkRmKN8UGbDjYfLxVFcB4T
['ABC']         Foc7XKHuZGwMt692zx4ReWg8Lmk1vQDpnrsi3EBbYPqhaNJA5TjCUyfSdV
['pascal']      i8myeM5bWSnw1kvKPGpxDh6cdoAg7TNLZfjH4tEJz2BCFuUVaQs93qYRXr

*)
```