sha512.cm

```
/*
    ========================================================================

    Copyright (c) 2012-2016 Seppo Laakko
    http://sourceforge.net/projects/cmajor/

    Distributed under the GNU General Public License, version 3 (GPLv3).
    (See accompanying LICENSE.txt or http://www.gnu.org/licenses/gpl.html
        )

    ========================================================================
    */

// Copyright (c) 1994
// Hewlett-Packard Company
// Copyright (c) 1996
// Silicon Graphics Computer Systems, Inc.
// Copyright (c) 2009 Alexander Stepanov and Paul McJones

using System.IO;

namespace System.Security
{
    public nothrow inline ulong RightRotate(ulong x, ulong n)
    {
        return (x >> n) | (x << (64u - n));
    }

    public class Sha512
    {
        static nothrow Sha512()
        {
            k[0] = 0x428a2f98d728ae22u;
            k[1] = 0x7137449123ef65cdu;
            k[2] = 0xb5c0fbcfec4d3b2fu;
            k[3] = 0xe9b5dba58189dbbcu;
            k[4] = 0x3956c25bf348b538u;
            k[5] = 0x59f111f1b605d019u;
            k[6] = 0x923f82a4af194f9bu;
            k[7] = 0xab1c5ed5da6d8118u;
            k[8] = 0xd807aa98a3030242u;
            k[9] = 0x12835b0145706fbeu;
            k[10] = 0x243185be4ee4b28cu;
            k[11] = 0x550c7dc3d5ffb4e2u;
            k[12] = 0x72be5d74f27b896fu;
            k[13] = 0x80deb1fe3b1696b1u;
            k[14] = 0x9bdc06a725c71235u;
            k[15] = 0xc19bf174cf692694u;
            k[16] = 0xe49b69c19ef14ad2u;
            k[17] = 0xefbe4786384f25e3u;
            k[18] = 0x0fc19dc68b8cd5b5u;
```

```
    k[19] = 0x240ca1cc77ac9c65u;
    k[20] = 0x2de92c6f592b0275u;
    k[21] = 0x4a7484aa6ea6e483u;
    k[22] = 0x5cb0a9dcbd41fbd4u;
    k[23] = 0x76f988da831153b5u;
    k[24] = 0x983e5152ee66dfabu;
    k[25] = 0xa831c66d2db43210u;
    k[26] = 0xb00327c898fb213fu;
    k[27] = 0xbf597fc7beef0ee4u;
    k[28] = 0xc6e00bf33da88fc2u;
    k[29] = 0xd5a79147930aa725u;
    k[30] = 0x06ca6351e003826fu;
    k[31] = 0x142929670a0e6e70u;
    k[32] = 0x27b70a8546d22ffcu;
    k[33] = 0x2e1b21385c26c926u;
    k[34] = 0x4d2c6dfc5ac42aedu;
    k[35] = 0x53380d139d95b3dfu;
    k[36] = 0x650a73548baf63deu;
    k[37] = 0x766a0abb3c77b2a8u;
    k[38] = 0x81c2c92e47edaee6u;
    k[39] = 0x92722c851482353bu;
    k[40] = 0xa2bfe8a14cf10364u;
    k[41] = 0xa81a664bbc423001u;
    k[42] = 0xc24b8b70d0f89791u;
    k[43] = 0xc76c51a30654be30u;
    k[44] = 0xd192e819d6ef5218u;
    k[45] = 0xd69906245565a910u;
    k[46] = 0xf40e35855771202au;
    k[47] = 0x106aa07032bbd1b8u;
    k[48] = 0x19a4c116b8d2d0c8u;
    k[49] = 0x1e376c085141ab53u;
    k[50] = 0x2748774cdf8eeb99u;
    k[51] = 0x34b0bcb5e19b48a8u;
    k[52] = 0x391c0cb3c5c95a63u;
    k[53] = 0x4ed8aa4ae3418acbu;
    k[54] = 0x5b9cca4f7763e373u;
    k[55] = 0x682e6ff3d6b2b8a3u;
    k[56] = 0x748f82ee5defb2fcu;
    k[57] = 0x78a5636f43172f60u;
    k[58] = 0x84c87814a1f0ab72u;
    k[59] = 0x8cc702081a6439ecu;
    k[60] = 0x90befffa23631e28u;
    k[61] = 0xa4506cebde82bde9u;
    k[62] = 0xbef9a3f7b2c67915u;
    k[63] = 0xc67178f2e372532bu;
    k[64] = 0xca273eceea26619cu;
    k[65] = 0xd186b8c721c0c207u;
    k[66] = 0xeada7dd6cde0eb1eu;
    k[67] = 0xf57d4f7fee6ed178u;
    k[68] = 0x06f067aa72176fbau;
    k[69] = 0x0a637dc5a2c898a6u;
    k[70] = 0x113f9804bef90daeu;
    k[71] = 0x1b710b35131c471bu;
```

```
        k[72] = 0x28db77f523047d84u;
        k[73] = 0x32caab7b40c72493u;
        k[74] = 0x3c9ebe0a15c9bebcu;
        k[75] = 0x431d67c49c100d4cu;
        k[76] = 0x4cc5d4becb3e42b6u;
        k[77] = 0x597f299cfc657e2au;
        k[78] = 0x5fcb6fab3ad6faecu;
        k[79] = 0x6c44198c4a475817u;
    }
    public nothrow  Sha512()
    {
        Reset();
    }
    public nothrow void Reset()
    {
        digest[0] = 0x6a09e667f3bcc908u;
        digest[1] = 0xbb67ae8584caa73bu;
        digest[2] = 0x3c6ef372fe94f82bu;
        digest[3] = 0xa54ff53a5f1d36f1u;
        digest[4] = 0x510e527fade682d1u;
        digest[5] = 0x9b05688c2b3e6c1fu;
        digest[6] = 0x1f83d9abfb41bd6bu;
        digest[7] = 0x5be0cd19137e2179u;
    }
    public nothrow void Process(byte x)
    {
        ProcessByte(x);
        bitCountLow = bitCountLow + 8u;
        if (bitCountLow == 0u)
        {
            ++bitCountHigh;
        }
    }
    public nothrow void Process(const void* begin, const void* end)
    {
        byte* b = cast<byte*>(begin);
        byte* e = cast<byte*>(end);
        while (b != e)
        {
            Process(*b);
            ++b;
        }
    }
    public nothrow void Process(const void* buf, int count)
    {
        byte* b = cast<byte*>(buf);
        Process(b, b + count);
    }
    public nothrow string GetDigest()
    {
        ProcessByte(0x80u);
        if (byteIndex > 112u)
        {
```

```
            while (byteIndex != 0u)
            {
                ProcessByte(0u);
            }
            while (byteIndex < 112u)
            {
                ProcessByte(0u);
            }
        }
        else
        {
            while (byteIndex < 112u)
            {
                ProcessByte(0u);
            }
        }
        ProcessByte(cast<byte>((bitCountHigh >> 56u) & 0xFFu));
        ProcessByte(cast<byte>((bitCountHigh >> 48u) & 0xFFu));
        ProcessByte(cast<byte>((bitCountHigh >> 40u) & 0xFFu));
        ProcessByte(cast<byte>((bitCountHigh >> 32u) & 0xFFu));
        ProcessByte(cast<byte>((bitCountHigh >> 24u) & 0xFFu));
        ProcessByte(cast<byte>((bitCountHigh >> 16u) & 0xFFu));
        ProcessByte(cast<byte>((bitCountHigh >> 8u) & 0xFFu));
        ProcessByte(cast<byte>(bitCountHigh & 0xFFu));
        ProcessByte(cast<byte>((bitCountLow >> 56u) & 0xFFu));
        ProcessByte(cast<byte>((bitCountLow >> 48u) & 0xFFu));
        ProcessByte(cast<byte>((bitCountLow >> 40u) & 0xFFu));
        ProcessByte(cast<byte>((bitCountLow >> 32u) & 0xFFu));
        ProcessByte(cast<byte>((bitCountLow >> 24u) & 0xFFu));
        ProcessByte(cast<byte>((bitCountLow >> 16u) & 0xFFu));
        ProcessByte(cast<byte>((bitCountLow >> 8u) & 0xFFu));
        ProcessByte(cast<byte>(bitCountLow & 0xFFu));
        string s = ToHexString(digest[0]);
        s.Append(ToHexString(digest[1]));
        s.Append(ToHexString(digest[2]));
        s.Append(ToHexString(digest[3]));
        s.Append(ToHexString(digest[4]));
        s.Append(ToHexString(digest[5]));
        s.Append(ToHexString(digest[6]));
        s.Append(ToHexString(digest[7]));
        return s;
    }
    private nothrow void ProcessByte(byte x)
    {
        block[byteIndex++] = x;
        if (byteIndex == 128u)
        {
            byteIndex = 0u;
            ProcessBlock();
        }
    }
    private nothrow void ProcessBlock()
    {
```

```
        ulong[80] w;
        for (int i = 0; i < 16; ++i)
        {
            w[i] = cast<ulong>(block[8 * i]) << 56u;
            w[i] = w[i] | cast<ulong>(block[8 * i + 1]) << 48u;
            w[i] = w[i] | cast<ulong>(block[8 * i + 2]) << 40u;
            w[i] = w[i] | cast<ulong>(block[8 * i + 3]) << 32u;
            w[i] = w[i] | cast<ulong>(block[8 * i + 4]) << 24u;
            w[i] = w[i] | cast<ulong>(block[8 * i + 5]) << 16u;
            w[i] = w[i] | cast<ulong>(block[8 * i + 6]) << 8u;
            w[i] = w[i] | cast<ulong>(block[8 * i + 7]);
        }
        for (int i = 16; i < 80; ++i)
        {
            w[i] = (RightRotate(w[i - 2], 19u) ^ RightRotate(w[i -
                2], 61u) ^ (w[i - 2] >> 6u)) + w[i - 7] + (RightRotate
                (w[i - 15], 1u) ^ RightRotate(w[i - 15], 8u) ^ (w[i -
                15] >> 7u)) + w[i - 16];
        }
        ulong a = digest[0];
        ulong b = digest[1];
        ulong c = digest[2];
        ulong d = digest[3];
        ulong e = digest[4];
        ulong f = digest[5];
        ulong g = digest[6];
        ulong h = digest[7];
        for (int i = 0; i < 80; ++i)
        {
            ulong t1 = h + (RightRotate(e, 14u) ^ RightRotate(e, 18u)
                ^ RightRotate(e, 41u)) + ((e & f) ^ ((~e) & g)) + k[i
                ] + w[i];
            ulong t2 = (RightRotate(a, 28u) ^ RightRotate(a, 34u) ^
                RightRotate(a, 39u)) + ((a & b) ^ (a & c) ^ (b & c));
            h = g;
            g = f;
            f = e;
            e = d + t1;
            d = c;
            c = b;
            b = a;
            a = t1 + t2;
        }
        digest[0] = a + digest[0];
        digest[1] = b + digest[1];
        digest[2] = c + digest[2];
        digest[3] = d + digest[3];
        digest[4] = e + digest[4];
        digest[5] = f + digest[5];
        digest[6] = g + digest[6];
        digest[7] = h + digest[7];
    }
    private static ulong[80] k;
```

```
        private byte[128] block;
        private byte byteIndex;
        private ulong bitCountLow;
        private ulong bitCountHigh;
        private ulong[8] digest;
    }

    public nothrow string GetSha512MessageDigest(const string& message)
    {
        Sha512 sha512;
        sha512.Process(message.Chars(), message.Length());
        return sha512.GetDigest();
    }

    public string GetSha512FileDigest(const string& filePath)
    {
        Sha512 sha512;
        BinaryFileStream file(filePath, OpenMode.readOnly);
        IOBuffer buffer(4096u);
        int bytesRead = file.Read(buffer.Mem(), buffer.Size());
        while (bytesRead > 0)
        {
            sha512.Process(buffer.Mem(), bytesRead);
            bytesRead = file.Read(buffer.Mem(), buffer.Size());
        }
        return sha512.GetDigest();
    }
}
```