

Forum: [Hack Forums](#)

Thread: [\[FREE SOURCE\] HC-128 stream cipher implementation for C#](#)

hgdfhsdgdgdfh - December 31, 2011, 9:43 am

I wrote this today.

It implements HC-128 stream cipher designed by Hongjun Wu.

HC-128 has been part of eSTREAM project (european project to "identify new stream ciphers suitable for widespread adoption").

See:

http://www.ecrypt.eu.org/stream/p3ciphers/hc/hc128_p3.pdf

<http://www.ecrypt.eu.org/stream/hcp3.html>

I have tested it on some test vectors but I cannot guarantee its 100% secure or that I implemented it correctly.

I think this is good alternative to RC4 cipher used commonly on these forums.

Code /*

* HC-128 stream cipher implementation by 0xDEADDEAD

* Code (not algorithm): Copyright (C) 2011 0xDEADDEAD

*/

using System;

using System.Security.Cryptography;

namespace AlgorithmTests

{

public sealed class HC128

{

// state

private UInt32[] _P;

private UInt32[] _Q;

private byte[] _key;

private byte[] _iv;

private UInt32 _count = 0;

/// <summary>

/// Initialize a new instance of HC-128 stream cipher using specified parameters

/// </summary>

/// <param name="key">Key (128 bits)</param>

/// <param name="iv">IV (128 bits)</param>

public HC128(byte[] key, byte[] iv)

{

_key = key;

_iv = iv;

}

public byte[] EncryptDecrypt(byte[] data)

```

{
    InitState();
    byte[] keyStream = new byte[data.Length + (data.Length % 4)];

    for (int i = 0 ; i < keyStream.Length / 4 ; i++)
    {
        byte[] keyB = BitConverter.GetBytes(GenerateKeyData());
        Buffer.BlockCopy(keyB, 0, keyStream, i * 4, 4);
    }

    byte[] result = new byte[data.Length];

    for (int i = 0; i < data.Length; i++)
    {
        result[i] = (byte) (data[i] ^ keyStream[i]);
    }

    return result;
}

```

```

/// <summary>
/// Initialize cipher state
/// </summary>
private void InitState()
{
    if (_key.Length != 16)
        throw new CryptographicException("invalid key size");

    if (_iv.Length != 16)
        throw new CryptographicException("invalid IV size");

    _count = 0;

    _P = new uint[512];
    _Q = new uint[512];
    UInt32[] KP = new uint[4];
    UInt32[] IVP = new uint[4];
    UInt32[] W = new uint[1280];

    KP[0] = BitConverter.ToUInt32(_key, 0);
    KP[1] = BitConverter.ToUInt32(_key, 4);
    KP[2] = BitConverter.ToUInt32(_key, 8);
    KP[3] = BitConverter.ToUInt32(_key, 12);

    IVP[0] = BitConverter.ToUInt32(_iv, 0);
    IVP[1] = BitConverter.ToUInt32(_iv, 4);
    IVP[2] = BitConverter.ToUInt32(_iv, 8);
    IVP[3] = BitConverter.ToUInt32(_iv, 12);

    for (int i = 0; i < 8; i++)
        W[i] = KP[i%4];

    for (int i = 0; i < 8; i++)

```

```

W[i + 8] = IVP[i % 4];

for (int i = 16; i < 1280; i++)
W[i] = (uint)(F2(W[i - 2]) + W[i - 7] + F1(W[i - 15]) + W[i - 16] + i);

for (int i = 0; i < 512; i++)
_P[i] = W[i + 256];

for (int i = 0; i < 512; i++)
_Q[i] = W[i + 768];

for (int i = 0; i < 512; i++)
_P[i] = (_P[i] +
G1(_P[MinMod512((uint) i, 3)], _P[MinMod512((uint) i, 10)], _P[MinMod512((uint) i, 511)]))
^ H1(_P[MinMod512((uint) i, 12)]);

for (int i = 0; i < 512; i++)
_Q[i] = (_Q[i] +
G2(_Q[MinMod512((uint)i, 3)], _Q[MinMod512((uint)i, 10)], _Q[MinMod512((uint)i, 511)]))
^ H2(_Q[MinMod512((uint)i, 12)]);

}

/// <summary>
/// Generate a key byte
/// </summary>
/// <returns>Generated dword</returns>
private UInt32 GenerateKeyData()
{
    uint j = _count%512;
    uint result = 0;
    if ((_count%1024) < 512)
    {
        _P[j] = _P[j] +
        G1(_P[MinMod512(j, 3)], _P[MinMod512(j, 10)], _P[MinMod512(j, 511)]);

        result = H1(_P[MinMod512(j, 12)]) ^ _P[j];
    }
    else
    {
        _Q[j] = _Q[j] +
        G2(_Q[MinMod512(j, 3)], _Q[MinMod512(j, 10)], _Q[MinMod512(j, 511)]);

        result = H2(_Q[MinMod512(j, 12)]) ^ _Q[j];
    }
    _count++;
    return result;
}

/// <summary>
/// a-b mod 512
/// </summary>
private UInt32 MinMod512(UInt32 a, UInt32 b)

```

```

{
    return (a - b)%512;
}

private UInt32 F1(UInt32 x)
{
    return RollRight(x, 7) ^ RollRight(x, 18) ^ (x >> 3);
}

private UInt32 F2(UInt32 x)
{
    return RollRight(x, 17) ^ RollRight(x, 19) ^ (x >> 10);
}

private UInt32 G1(UInt32 x, UInt32 y, UInt32 z)
{
    return (RollRight(x, 10) ^ RollRight(z, 23)) + RollRight(y, 8);
}

private UInt32 G2(UInt32 x, UInt32 y, UInt32 z)
{
    return (RollLeft(x, 10) ^ RollLeft(z, 23)) + RollLeft(y, 8);
}

private UInt32 H1(UInt32 x)
{
    return _Q[(byte)x] + _Q[256 + (byte)(x >> 16)];
}

private UInt32 H2(UInt32 x)
{
    return _P[(byte)x] + _P[256 + (byte)(x >> 16)];
}

private UInt32 RollRight(UInt32 x, int n)
{
    return ((x >> n) ^ (x << (32 - n)));
}

private UInt32 RollLeft(UInt32 x, int n)
{
    return ((x << n) ^ (x >> (32 - n)));
}
}

```

xSilent - December 31, 2011, 12:11 pm

?

An encryption system in .NET that doesn't use a CryptoStream?! Nani?! (Japanese for What?!)

Hella nice job! ?

pac - January 1, 2012, 4:20 am

I absolutely love your threads! I would +100 you if I had the powers to do it!

hgdfhsdgdgdfh - January 1, 2012, 9:24 am

Thanks.

I don't like inheriting my classes from .NET base classes at least when it comes to cryptography :P
It just makes most things harder.