Crypto Forum Research Group
Internet-Draft
Intended status: Informational

N. Huckleberry
Google

August 19, 2021

HCTR Specification

Abstract

This memo describes a block cipher mode of operation known as HCTR.

Status of This Memo

Copyright Notice

Table of Contents

1 Introduction

This memo describes the implementation of the HCTR block cipher mode of operation. The primary benefit of HCTR is that it extends the strong pseudorandom permutation property of block ciphers to arbitrary-length messages. Additionally, HCTR allows for an arbitrary-length nonce called a tweak.

1.1 Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2 Notation

block - 16 bytes.

block_cipher_encrypt(msg, key) - a subroutine call to the underlying blockcipher encryption function.

block_cipher_decrypt(msg, key) - a subroutine call to the underlying blockcipher decryption function.

block_key - key for the underlying ecb block cipher calls.

buffer[i] - block i of buffer. Defined for 0 <= i < N.

buffer[N+] - bytes 16 * N until the end of buffer.

EMPTY - buffer of length zero.

GF(2^128) - Galois field of order 2^128 as described in section 4.

hash_key - key for the polynomial hash function.

hctr_key - full key for HCTR, contains block_key and hash_key.

msg - shorthand for message, a buffer that is an input to a function.

TRUNC(X, n) - truncate X to be n bits.

XXXX_length - length of XXXX in bytes.

+ - addition of integers

# - addition in GF(2^128) as defined in section 4.

* - multiplication in GF(2^128) as defined in section 4.

0^i - buffer of i zero bytes.

|| - concatenation.

|x| - length of x in bits.

x || 0^* - x right-padded with 0 to be a multiple of 16 bytes.

3 Overview

3.1 Key size

The HCTR key consists of two parts, the underlying block cipher key and the hash key. The hash key is always 128 bits, so the full HCTR key size in bytes is block_key_length + 16.

3.2 Block Cipher

HCTR MUST use a block cipher with a block size of 128-bits.

4 GF(2^128) Math

4.1 GF(2^128)

GF(2^128) is the Galois field with 2^128 elements defined by the irreducible polynomial x^128 + x^7 + x^2 + x + 1.

Elements in the field are converted to and from 128-bit strings by taking the least significant bit of the first byte to be the coefficient of x^0, the most significant bit of the first byte to be the coefficient of x^7, and so on, until the most significant bit of the last byte is the coefficient of x^127.

4.2 Addition in 2^128

Input

Two 128-bit elements X, Y

Ouput

128-bit element X # Y

For any two 128-bit elements X, Y in the Galois field, X # Y is defined as X XOR Y.

The operations + and XOR are interchangeable within this document. For consistency we use + on 128-bit strings and XOR if the arguments are not 128-bits long.

4.3 Multiplication in 2^128

Input

Two 128-bit elements X, Y

Ouput

128-bit element X * Y

Multiplication is defined on 128-bit blocks by converting them to polynomials as described above, and then computing the resulting product modulo x^128 + x^7 + x^2 + x + 1.

5 Algorithm

When appropriate, we will explain the output as both a mathematical formula and in pseudo-code. This information is redundant, and it exists to provide additional clarity. Implementations need not implement the exact algorithm specified by the pseudocode, so long as the output matches what the pseudocode would produce.

5.1 Keys

The HCTR key is a byte string of length block_key_length + 16. The polynomial hash key and underlying block cipher key are the respective substrings of the HCTR key.

```
    hctr_key = hash_key || block_key
```

5.2 poly_hash

The polynomial hash treats each block of msg as a coefficient to a polynomial in GF(2^128), and evaluates that polynomial at hash_key to create a hash.

This hash function dominates the efficiency of HCTR, special care should be taken to optimize the efficiency of poly_hash. In particular, multiplication in GF(2^128) is expensive, so pre-computing powers of hash_key can signficantly increase HCTR’s efficiency.

```
    Input

        msg, hash_key

    Output

        if (msg == EMPTY)

            k

        if (msg != EMPTY)

            k^(N+1)*m_0 # k^N*m_1 # ... # k^2*m_{N-1} # |msg|*k

        Where k = hash_key,

        k^i is the ith power of k in GF(2^128)

        m = msg || 0^*

        N = number of blocks in m

        m_i = m[i] for 0 <= i <= N-1

    Pseudo-code

        Precompute k^2 ... k^33

        if (N == 0)

            return k

        p = 0^16

        for i = 0 to (N/32)-1

            p = p * k^32

            for j = 0 to 31
            p = p # (m[i*32 + j] * k^(33 - j))
        x = N % 32
        p = p * k^x
        for i = 0 to x-1
            p = p # (m[(N/32)*32 + i] * k^(x + 1 - i))
        return p
```

5.3 HCTR_ctr_crypt

HCTR uses a modified version of CTR as a subroutine. The modified CTR requires a 128-bit initialization vector and produces a ciphertext with the same bit-length as the input message.

This modified CTR mode is an involution enciphering scheme, meaning that the encryption function is the same as the decryption function. Furthermore, it can be easily parallelized.

```
    Input
        msg, block_key, iv
    Output
        out_msg = c_0 || ... || c_{m-2} || c_final
        Where m = msg || 0^*
        N = number of blocks in m
        m_i = m[i]
        c_i = m_i XOR block_cipher_encrypt(iv XOR (i + 1), block_key)
        c_final = TRUNC(c_{N-1}, |msg| % 256)
    Pseudo-code:
        out_msg = 0^|msg|
        for i = 0 to N-2
            tmp = block_cipher_encrypt(iv XOR (i + 1), block_key)
            out_msg[i] = m[i] XOR tmp
        final = block_cipher_encrypt(iv XOR N, block_key)
        final = TRUNC(final, |msg| % 256)
        out_msg[N-1] = final XOR m[N-1]
```

5.4 HCTR_encrypt

Core encryption function of HCTR. The length of the input message MUST be at least block_size. The tweak can be any length, including zero, but the tweak length SHOULD stay constant between HCTR calls.

```
Input

    msg, key, tweak

Output

    out_msg

Pseudo-code:

    M = msg[0]

    N = msg[1+]

    MM = M XOR poly_hash(N || T, hash_key)

    CC = block_cipher_encrypt(MM, block_key)

    S = MM XOR CC

    D = HCTR_ctr_crypt(N, block_key, S)

    C = CC XOR poly_hash(D || T, hash_key)

    out_msg = C || D
```

5.5 HCTR_decrypt

Core decryption function of HCTR. The length of the input message MUST be at least block_size. The tweak can be any length, including zero, but the tweak length SHOULD stay constant between HCTR calls.

```
Input

    msg, key, tweak

Output

    out_msg

Pseudo-code:

    C = msg[0]

    D = msg[1+]

    CC = C XOR poly_hash(D || T, hash_key)

    MM = block_cipher_decrypt(CC, block_key)

    S = MM XOR CC
```

```
        N = HCTR_ctr_crypt(D, block_key, S)
        M = MM XOR poly_hash(N || T, hash_key)
        out_msg = M || N
```

6 Security Considerations

The minimum length of the plaintext for HCTR is 16 bytes. The maximum length is 2^32 - 1 bytes.

6.1 Security Implications of Tweak Use

If no tweak is used (or, equivalently, if a tweak is reused for multiple messages) then HCTR is a strong pseudorandom permutation. If the same plaintext is encrypted twice with no tweak (or the same tweak), the resultant ciphertexts will match.

If a unique tweak is used for every plaintext and key combination, then HCTR is semantically secure. We make no claims that using randomly generated tweaks or longer tweaks generates additional security.

7 References

7.1 Normative References

[RFC2119] S. Bradner. “Key words for use in RFCs to Indicate Requirement Levels”. BCP14, RFC2119 (1997). DOI: 10.17487/RFC2119.

7.2 Informative References

[AES-GCM-SIV] A. Langley S. Gueron and Y. Lindell. “AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption.” (2016).

[CN08] Debrup Chakraborty and Mridul Nandi. “An Improved Security Bound for HCTR” (2008).

[HCTR] Peng Wang, Dengguo Feng, and Wenling Wu. “HCTR: A Variable-Input-Length Enciphering Mode” (2005).

[HEH] A. Cope. “Hash-Encrypt-Hash, a block cipher mode of operation” (2016).

Appendix A Test Vectors