Zcipher Algorithm Specification

Ilya O. Levin, eli@literatecode.com

This note describes the Zcipher algorithm. It is a symmetric encryption/decryption cryptographic algorithm, designed to be secure and high-performance with small memory footprint and simple structure.

1. Algorithm Description

Zcipher is a 64-bit codebook, parameterized by a 128-bit cryptovariable. The codebook is a Feistel network, where 64-bit input data splits in two 32-bit values and 128-bit cryptovariable splits in four 32-bit values.

1.1 Encryption

Figure 1 demonstrates a single encryption round; {*x*, *y*} is an input data and $v_{i=1,2,3,4}$ is a cryptovariable, all arithmetic operations are mod 2^{32}.

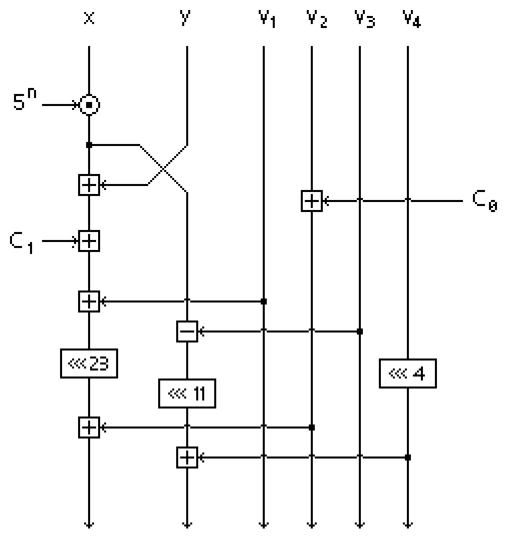

Figure 1: "A single round of encryption"

A full period single cycle linear congruential generator $f(x) \rightarrow 5^n x + C$ is using as an invertible mapping for both fixed and cryptovariable-dependent data substitution. Exponent *n* can be any natural number. If $n = 1$ then the generator can be implemented with LEA instruction in just a single CPU cycle on Intel Pentium and above. The recommended *n* for Zcipher is 13; it gives a maximal 5^n value less than 2^{32}.

Zcipher requires a minimum of 8 rounds. More rounds may be used, but notice that having number of rounds as multiple 8 simplify decrypt preface on v_4. After the last round the transformed data combined with cryptovariable by XOR:

$$x \rightarrow x \oplus v_3$$
$$y \rightarrow y \oplus v_1$$

1.2 Decryption

The inverse operations must be applied in reverse order to decrypt the encrypted data.

1.3 Cryptovariable schedule

The schedule is a pseudo random generator with 128-bit output. The output is a cryptovariable. The generator is a 128-bit state one without output filter; the state is an output.

The generator based on a function

$$f(a, b, r, C) \rightarrow (((a - b) + C)_{<<<\,19} + b)_{<<<\,r}$$

where C is an odd constant. Assuming $v_{i=1,2,3,4}$ is a state, the schedule is 4 rounds of the following transformation:

$$\begin{aligned} v_1 &:= f(v_2, v_1, 11, C_0) \\ v_2 &:= f(v_3, v_2, 9, C_1) \\ v_3 &:= f(v_4, v_3, 7, C_2) \\ v_4 &:= f(v_1, v_4, 10, C_3) \end{aligned}$$

The constants C_i are

$$\begin{aligned} C_0 &= 9E3779B9_{hex} \\ C_1 &= E2E4C7C5_{hex} \\ C_2 &= 16C7D03B_{hex} \\ C_3 &= 3A11584F_{hex} \end{aligned}$$

Constants C_2 and C_3 are primes; C_0 is the golden ratio constant and C_1 is an obvious reference to the Sicilian defense.

Four rounds is a minimum that sufficient to generate an output able to pass statistical tests on randomness. Either any 32-bit v or the whole 128-bit state taken as output successfully passes various such tests.

2. Modes of Operation

Although operating modes defined in FIPS-81 [1] are employable, Zcipher primarily designed for a stream mode, where cryptovariable scheduled before processing for each data block. That is the encryption and decryption routines for data M_i and cryptovariable V_i are

```
procedure Zcipher_Encrypt($M_i$, $V_i$)
{
   $V_{i+1}$:= Schedule($V_i$)
   $C_i$:= Encrypt($M_i$, $V_{i+1}$)
   return $C_i$
}
```

```
procedure Zcipher_Decrypt($M_i$, $V_i$)
{
   $V_{i+1}$:= Schedule($V_i$)
   $C_i$:= Decrypt($M_i$, $V_{i+1}$)
   return $C_i$
}
```

To some extent, Zciper may be seen as a quadruple-word oriented stream cipher with a codebook as an output filter function to scheduling PRNG.

Single cryptovariable stream can be shared between both encryption and decryption routines during online sessions. Since cryptovariable is data-independent, it can be scheduled in parallel and pre-computed few blocks ahead.

3. Security Considerations

The design of Zcipher is heavily affected by tradeoffs. It is aggressively simplified in favor of small footprint, light structure and performance. Thus Zcipher might be close to a margin. Any further simplification such as reduced rounds, removed final XOR, etc. will obviously break the cipher.

Use invertible mapping with LCG instead of a traditional S-box transformation might be a weak point.

Fixed rotation with addition mod 2^{32} might be a good point to try the *mod n* attack [2].

The codebook size might be considered too small for a generic block cipher by modern standards.

Summarizing all above, Zcipher is an experimental cipher for research purposes. It must be taken with care and avoided in real life systems where security is critical.

4. References

1. FIPS PUB 81. DES MODES OF OPERATIONS
   http://www.itl.nist.gov/fipspubs/fip81.htm

2. J. Kelsey, B. Schneier, and D. Wagner: Mod *n* Cryptanalysis, with Applications against RC5P and M6. Fast Software Encryption, Sixth International Workshop Proceedings (March 1999), Springer-Verlag, 1999, pp. 139-155.
   http://www.schneier.com/paper-mod3.html

APPENDIX A. Test Vectors

Cryptovariable Schedule

```
INITIAL VALUE:      11111111 22222222 33333333 44444444
SCHEDULED OUTPUT 1: 7a7db236 8d69797c ea87711e e183da90
SCHEDULED OUTPUT 2: bd03c8dc 98e5347f 8cfb9730 45dc0cc9
SCHEDULED OUTPUT 3: 96fbbf1d 2f7d4382 c3fabc58 42f34f4d
```

Codebook (8 rounds, *n* = 13)

```
PLAINTEXT:  12345678 9abcdef0
CIPHERTEXT: debad3c0 eebfe559

PLAINTEXT:  00000000 00000000
CIPHERTEXT: bea03c93 3d9ea4c2

PLAINTEXT:  ffffffff ffffffff
CIPHERTEXT: 21c18851 c3f80dd5
```

APPENDIX B. Source Code in C

```
/*
*   Zcipher. Reference implementation in C
*   Written by Ilya O. Levin, http://www.literatecode.com
*/

#define uint32_t  unsigned long
#define R(x,y)     (((x)<<(y))|((x)>>(32-(y))))

#define C0       0x9e3779b9
#define C1       0xE2E4C7C5
#define C2       0x16C7D03B
#define C3       0x3A11584F

#define ROUNDS   8
#define SB5      0x48C27395
#define SB5inv   0xF6433FBD

#define F(a,b, c, x, C) ( R(R((a - b) + C, x) + b, c) )


void cvsched(uint32_t *v)
{
    register uint32_t i = ROUNDS / 2;

    while (i-->0)
    {
        v[2] = F(v[3], v[2], 11,19, C0);
        v[3] = F(v[4], v[3], 9, 19, C1);
        v[4] = F(v[5], v[4], 7, 19, C2);
        v[5] = F(v[2], v[5], 10,19, C3);
    }

} /* cvsched */
```

```c
void encr(uint32_t *v)
{
    uint32_t t, x = v[0], y = v[1],
            a = v[2], b = v[3], c = v[4], d = v[5], i = ROUNDS;

    while (i-->0)
    {
        t = x * SB5;
        x = y + C1 + t;
        y = t;
        b += C0;
        d = R(d, 4);
        x = R(x + a, 23) + b;
        y = R(y - c, 11) + d;
    }
    v[0] = x ^ c; v[1] = y ^ a;

} /* encr */



void decr(uint32_t *v)
{
    uint32_t t, x = v[0], y = v[1],
            a = v[2], b = v[3], c = v[4], d = v[5], i = ROUNDS;

    x ^= c; y ^= a;
    b += ((ROUNDS * C0) & 0xFFFFFFFF);

    while (i-->0)
    {
        y = R(y - d, 21) + c;
        x = R(x - b, 9) - a;
        b -= C0;
        d = R(d, 28);
        x -= y;
        t = y * SB5inv;
        y = x - C1;
        x = t;
    }
    v[0] = x; v[1] = y;

} /* decr */
```