Spock Block Cipher

Karl Zander – pvial00@gmail.com

Abstract.

Spock is an encryption algorithm that obfuscates blocks of data so that no one is able to read it without the key. Based on the Speck cipher designed by the NSA, it is an ARX cipher using only addition mod 2^64, bitwise rotations and the XOR operator. It has a 128 bit block size and uses more rounds than Speck.

1. **Introduction**

Spock is a fast block cipher operating on 128 bits of data at a time.

Spock consists of a key schedulers, one for the round keys and one for the substitution boxes and a round function. It uses substitution and ARX operations to encrypt data and operates on blocks of 128 bits of data.

The cipher is designed to operate at key lengths from 128 bits to 256 bits. An initialization vector is required for encryption. The length of the IV must be 128-bits to match the block size (could be be different depending on the mode in implementation).

2. **Design goals**

The algorithm must be comparably fast to the Speck cipher and have more rounds than the Speck cipher.

The algorithm must adhere to the strict avalanche effect and produce proper confusion and diffusion.

The algorithm must employ an initialization vector that does not effect the key generation process.

The algorithm must produce a non-repeating stream (without period) of bytes with uniform characteristics.

The algorithm must pass known statistical testing for random number generators. The algorithm must be fast to implement in software.

3. **Round Key Scheduling**

There are 192 bits of key material per round applied to the block. There are two 32 bit primary round keys and four 32 bit diffusion keys.

The key scheduler is very simple. It loads the key into a 64 bit word array k[], creates a temporary copy of it's key state, then loads the key k[] into that state. The key array is then run through Spock's round function to produce a new key each time it's run that is loaded into the primary key state.

This is done in the following order, Ka[] is generated, Kb[] is generated and finally the diffusion keys are generated.

For 128 bit keys, the round function is run once for every key, for 256 bit keys it's necessary it run the round function twice, once for each half of the key array.

5. **Round Encryption/Decryption**

Only ARX operations are used during the round. The cipher operates on 128 bits of data at a time split up into four 64 bit words. The words are subjected one of many functions such as: being bitwise rotated left or right so many positions, added to other words modulo 2^64, XOR'ed with a primary round key, XOR'ed with another word and being added to the diffusion key.

In code and in this example of one round of encryption we'll refer to those four words as A, B, C, and D in that order. Encryption is done in 3 phases.

Encryption:

1. ARX phase
    - First, A is bitwise rotated right 8 positions, A is then added to D, A is XOR'ed with a primary key from array Ka[].
    - Second, B is rotated right 7 positions, B is added to C, B is XOR'ed with a primary key from array Kb[].
    - Third, C is rotated left 2 positions, C is XOR'ed with B.
    - Fourth, D is rotated left 3 positions, D is XOR'ed with A.

2. Cross addition phase
    - In this phase A is simply added to B and B to A.
3. Diffusion key phase
    - The four diffusion keys are added to A, B, C and D.

Decryption:

1. Diffusion key phase
    - A, B, C and D are subtracted from the four diffusion keys.

2. Cross subtraction phase
    - In this phase B is simply subtracted from A and A from B.
3. ARX phase
    - First, D is XOR'ed with A and then D is rotated right 3 positions.
    - Second, C is XOR'ed with B and then C is rotated right 2 positions.
    - Third, B is XOR'ed with a primary key from array Kb[]. Then B is subtracted from C. B is then rotated left 7 positions.
    - Fourth, A is XOR'ed with a primary key from array Ka[]. Then A is subtracted from D. Finally, A is rotated left 8 positions.

The following code demonstrates both the encryption and decryption functions:

Encryption:

```
void roundF(struct spock_state *state, uint32_t *xla, uint32_t *xlb, uint32_t *xra, uint32_t *xrb, int rounds) {
   uint32_t a, b, c, d;
   a = *xla;
   b = *xlb;
   c = *xra;
   d = *xrb;
   for (int r = 0; r < rounds; r++) {
      a = spock_rotr(a, 8);
      a += d;
      a ^= state->Ka[r];
      b = spock_rotr(b, 7);
      b += c;
      b ^= state->Kb[r];
      c = spock_rotl(c, 2);
      c ^= b;
      d = spock_rotl(d, 3);
      d ^= a;
      a += b;
      b += a;
      a += state->d[r][0];
      b += state->d[r][1];
      c += state->d[r][2];
      d += state->d[r][3];
   }
   *xla = a;
   *xlb = b;
   *xra = c;
   *xrb = d;
}
```

Decryption:

```
void roundB(struct spock_state *state, uint32_t *xla, uint32_t *xlb, uint32_t *xra, uint32_t *xrb, int rounds) {
   uint32_t a, b, c, d;
   a = *xla;
   b = *xlb;
   c = *xra;
   d = *xrb;
   for (int r = rounds; r --> 0;) {
      d -= state->d[r][3];
      c -= state->d[r][2];
      b -= state->d[r][1];
      a -= state->d[r][0];
      b -= a;
      a -= b;
      d ^= a;
      d = spock_rotr(d, 3);
      c ^= b;
      c = spock_rotr(c, 2);
      b ^= state->Kb[r];
      b -= c;
      b = spock_rotl(b, 7);
      a ^= state->Ka[r];
      a -= d;
      a = spock_rotl(a, 8);
   }
   *xla = a;
   *xlb = b;
   *xra = c;
   *xrb = d;
}
```

6. **Cryptanalysis**

TBD

7. **Statistical Properties**

Spock was subjected to the Diehard battery of tests (dieharder). Overall, Spock passed the tests. Tests were conducted on streams of 1 gigabyte of data with 100 different key, nonce pairs. Spock was also subjected to NIST Statistical Test Suite battery of tests and passed.