

# The stream cipher MICKEY 2.0

**Steve Babbage**

**Matthew Dodd**

**Vodafone Group R&D, Newbury, UK**

steve.babbage@vodafone.com

**Independent consultant**

matthew@mdodd.net

www.mdodd.net

**30<sup>th</sup> June 2006**

**Abstract:** We present a strengthened version 2.0 of the stream cipher MICKEY. MICKEY (which stands for Mutual Irregular Clocking KEYstream generator) is aimed at resource-constrained hardware platforms. It is intended to have low complexity in hardware, while providing a high level of security. It uses irregular clocking of shift registers, with some novel techniques to balance the need for guarantees on period and pseudorandomness against the need to avoid certain cryptanalytic attacks.

**Keywords:** MICKEY, stream cipher, ECRYPT, irregular clocking.

## 1. Introduction

We present the stream cipher MICKEY 2.0 (which stands for Mutual Irregular Clocking KEYstream generator).

MICKEY 2.0 is aimed at resource-constrained hardware platforms. It is intended to have low complexity in hardware, while providing a high level of security.

The changes from MICKEY version 1, and the rationale behind them, are explained in section 8.

## 2. Input and output parameters

MICKEY 2.0 takes two input parameters:

- an 80-bit secret key  $K$ , whose bits are labelled  $k_0 \dots k_{79}$ ;
- an initialisation variable  $IV$ , anywhere between 0 and 80 bits in length, whose bits are labelled  $iv_0 \dots iv_{IVLENGTH-1}$ .

The keystream bits output by MICKEY 2.0 are labelled  $z_0, z_1, \dots$ . Ciphertext is produced from plaintext by bitwise XOR with keystream bits, as in most stream ciphers.

## 3. Acceptable use

The maximum length of keystream sequence that may be generated with a single  $(K, IV)$  pair is  $2^{40}$  bits. It is acceptable to generate  $2^{40}$  such sequences, all from the same  $K$  but with different values of  $IV$ . It is not acceptable to use two initialisation variables of different

lengths with the same  $K$ . And it is not, of course, acceptable to reuse the same value of  $IV$  with the same  $K$ .

## 4. Components of the keystream generator

### 4.1 The registers

The generator is built from two registers  $R$  and  $S$ . Each register is 100 stages long, each stage containing one bit. We label the bits in the registers  $r_0 \dots r_{99}$  and  $s_0 \dots s_{99}$  respectively.

Broadly speaking, we think of  $R$  as “the linear register” and  $S$  as “the non-linear register”.

### 4.2 Clocking the register $R$

Define a set of feedback tap positions for  $R$ :

$$RTAPS = \{0,1,3,4,5,6,9,12,13,16,19,20,21,22,25,28,37,38,41,42,45,46,50,52,54,56,58,60,61,63,64,65,66,67,71,72,79,80,81,82,87,88,89,90,91,92,94,95,96,97\}$$

We define an operation  $CLOCK\_R(R, INPUT\_BIT\_R, CONTROL\_BIT\_R)$  as follows:

- Let  $r_0 \dots r_{99}$  be the state of the register  $R$  before clocking, and let  $r'_0 \dots r'_{99}$  be the state of the register  $R$  after clocking.
- $FEEDBACK\_BIT = r_{99} \oplus INPUT\_BIT\_R$
- For  $1 \leq i \leq 99$ ,  $r'_i = r_{i-1}$ ;  $r'_0 = 0$
- For  $0 \leq i \leq 99$ , if  $i \in RTAPS$ ,  $r'_i = r'_i \oplus FEEDBACK\_BIT$
- If  $CONTROL\_BIT\_R = 1$ :
  - For  $0 \leq i \leq 99$ ,  $r'_i = r'_i \oplus r_i$

### 4.3 Clocking the register $S$

Define four sequences  $COMPO_1 \dots COMPO_{98}$ ,  $COMP1_1 \dots COMP1_{98}$ ,  $FBO_0 \dots FBO_{99}$ ,  $FB1_0 \dots FB1_{99}$  as follows:

$i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
$COMPO_i$		0	0	0	1	1	0	0	0	1	0	1	1	1	1	0	1	0	0	1	0	1	0	1	0
$COMP1_i$		1	0	1	1	0	0	1	0	1	1	1	1	0	0	1	0	1	0	0	0	1	1	0	1
$FBO_i$	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1
$FB1_i$	1	1	1	0	1	1	1	0	0	0	0	1	1	1	0	1	0	0	1	1	0	0	0	1	0
$i$	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
$COMPO_i$	1	0	1	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	1
$COMP1_i$	0	1	1	1	0	1	1	1	1	0	0	0	1	1	0	1	0	1	1	1	0	0	0	0	1
$FBO_i$	1	1	1	1	0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	1	0	0	1	0	1
$FB1_i$	0	1	1	0	0	1	0	1	1	0	0	0	1	1	0	0	0	0	0	1	1	0	1	1	0

<i>i</i>	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
<i>COMPO<sub>i</sub></i>	0	0	0	0	1	0	1	0	0	1	1	1	1	0	0	1	0	1	0	1	1	1	1	1	1
<i>COMP1<sub>i</sub></i>	0	0	0	1	0	1	1	1	0	0	0	1	1	1	1	1	1	0	1	0	1	1	1	0	1
<i>FBO<sub>i</sub></i>	0	1	0	0	1	0	1	1	1	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
<i>FB1<sub>i</sub></i>	0	0	1	0	0	0	1	0	0	1	0	0	1	0	1	1	0	1	0	1	0	0	1	0	1
<i>i</i>	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
<i>COMPO<sub>i</sub></i>	1	1	1	0	1	0	1	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	1	
<i>COMP1<sub>i</sub></i>	1	1	1	0	0	0	1	0	0	0	0	1	1	1	0	0	0	1	0	0	1	1	0	0	
<i>FBO<sub>i</sub></i>	1	1	0	1	0	0	0	1	1	0	1	1	1	0	0	1	1	1	0	0	1	1	0	0	0
<i>FB1<sub>i</sub></i>	0	0	0	1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	1

We define an operation  $\text{CLOCK\_S}(S, \text{INPUT\_BIT\_S}, \text{CONTROL\_BIT\_S})$  as follows:

- Let  $s_0 \dots s_{99}$  be the state of the register  $S$  before clocking, and let  $s'_0 \dots s'_{99}$  be the state of the register after clocking. We will also use  $\hat{s}_0 \dots \hat{s}_{99}$  as intermediate variables to simplify the specification.
- $\text{FEEDBACK\_BIT} = s_{99} \oplus \text{INPUT\_BIT\_S}$
- For  $1 \leq i \leq 98$ ,  $\hat{s}_i = s_{i-1} \oplus ((s_i \oplus \text{COMPO}_i) \cdot (s_{i+1} \oplus \text{COMP1}_i))$ ;  $\hat{s}_0 = 0$ ;  $\hat{s}_{99} = s_{98}$ .
- If  $\text{CONTROL\_BIT\_S} = 0$ :
  - For  $0 \leq i \leq 99$ ,  $s'_i = \hat{s}_i \oplus (\text{FBO}_i \cdot \text{FEEDBACK\_BIT})$
- If instead  $\text{CONTROL\_BIT\_S} = 1$ :
  - For  $0 \leq i \leq 99$ ,  $s'_i = \hat{s}_i \oplus (\text{FB1}_i \cdot \text{FEEDBACK\_BIT})$

#### 4.4 Clocking the overall generator

We define an operation  $\text{CLOCK\_KG}(R, S, \text{MIXING}, \text{INPUT\_BIT})$  as follows:

- $\text{CONTROL\_BIT\_R} = s_{34} \oplus r_{67}$
- $\text{CONTROL\_BIT\_S} = s_{67} \oplus r_{33}$
- If  $\text{MIXING} = \text{TRUE}$ , then  $\text{INPUT\_BIT\_R} = \text{INPUT\_BIT} \oplus s_{50}$ ; if instead  $\text{MIXING} = \text{FALSE}$ , then  $\text{INPUT\_BIT\_R} = \text{INPUT\_BIT}$
- $\text{INPUT\_BIT\_S} = \text{INPUT\_BIT}$
- $\text{CLOCK\_R}(R, \text{INPUT\_BIT\_R}, \text{CONTROL\_BIT\_R})$
- $\text{CLOCK\_S}(S, \text{INPUT\_BIT\_S}, \text{CONTROL\_BIT\_S})$

## 5. Key loading and initialisation

The registers are initialised from the input variables as follows:

- Initialise the registers  $R$  and  $S$  with all zeros.

- (Load in  $IV$ .) For  $0 \leq i \leq IVLENGTH - 1$ :
  - $CLOCK\_KG(R, S, MIXING = TRUE, INPUT\_BIT = iv_i)$
- (Load in  $K$ .) For  $0 \leq i \leq 79$ :
  - $CLOCK\_KG(R, S, MIXING = TRUE, INPUT\_BIT = k_i)$
- (Preclock.) For  $0 \leq i \leq 99$ :
  - $CLOCK\_KG(R, S, MIXING = TRUE, INPUT\_BIT = 0)$

## 6. Generating keystream

Having loaded and initialised the registers, we generate keystream bits  $z_0 \dots z_{L-1}$  as follows:

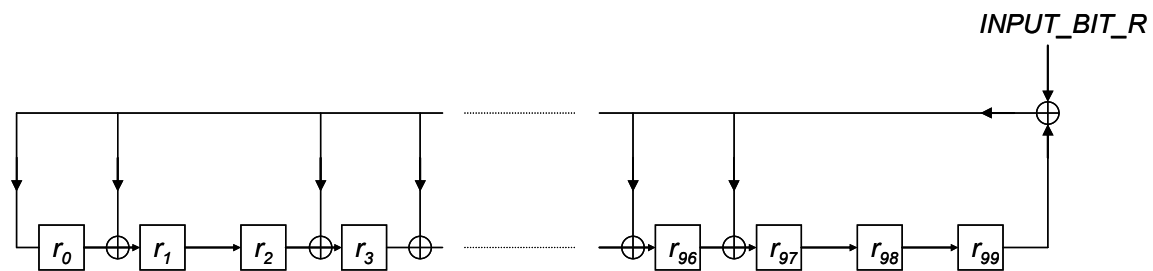
- For  $0 \leq i \leq L - 1$ :
  - $z_i = r_0 \oplus s_0$
  - $CLOCK\_KG(R, S, MIXING = FALSE, INPUT\_BIT = 0)$

## 7. Design principles

### 7.1 The variable clocking of $R$ : what it does

When  $CONTROL\_BIT\_R = 0$ , the clocking of  $R$  is a standard linear feedback shift register clocking operation (with Galois-style feedback, following the primitive characteristic polynomial  $C_R(x) = x^{100} + \sum_{i \in RTAPS} x^i$ , with  $INPUT\_BIT\_R$  XORed into the feedback).

If we represent elements of the field  $GF(2^{100})$  as polynomials  $\sum_{i=0}^{99} r_i x^i$ , modulo  $C_R(x)$ , then shifting the register corresponds to multiplication by  $x$  in the field.



**Figure 1:** Clocking the  $R$  register with  $CONTROL\_BIT\_R = 0$

When  $CONTROL\_BIT = 1$ , as well as shifting each bit in the register to the right, we also XOR it back into the current stage, as shown in Figure 2. This corresponds to multiplication by  $x + 1$  in the same field.

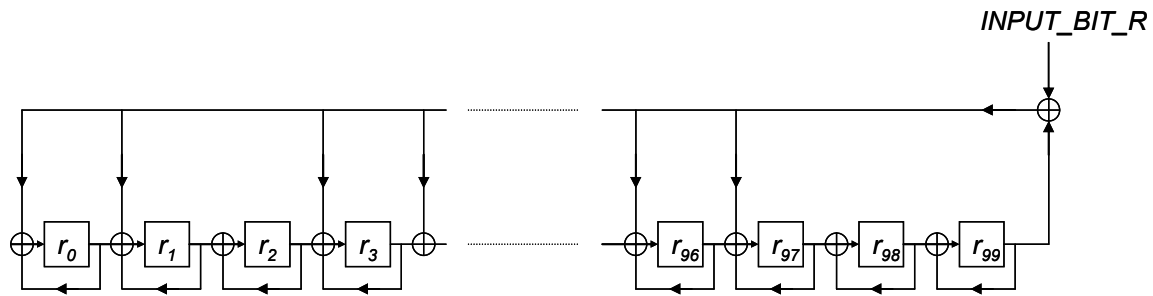


Figure 2: Clocking the  $R$  register with  $CONTROL\_BIT\_R = 1$

The characteristic polynomial  $C_R(x)$  has been chosen so that  $C_R(x) \mid x^J + x + 1$ , where  $J = 2^{50} - 157$ . Thus, clocking the register with  $CONTROL\_BIT\_R = 1$  is equivalent to clocking the register  $J$  times.

This technique — a simple operation, related to the standard linear register clocking operation but equivalent to making the register “jump” by clocking it  $J$  times — is due to Cees Jansen [8]. In [8], Jansen presents the technique applied to LFSRs with Fibonacci-style clocking, but it is clear that the same approach is valid with Galois-style clocking.

## 7.2 Motivation for the variable clocking

Stream ciphers making use of variable clocking often lend themselves to statistical attacks, in which the attacker guesses how many times the register has been clocked at a particular time. There are a number of characteristics of a cipher design that may make such attacks possible.

To illustrate these possible characteristics, let us consider the stream cipher LILI-128 [4]. LILI-128 uses two LFSRs, of length 39 and 89; the 89-stage register is clocked 1, 2, 3 or 4 times at each clock of the overall generator, based on two control bits from the 39-stage register. Attacks based on guessing a likely number of clocks of the 89-stage register may be possible because:

- (a) Clocking the 89-stage register  $m$  times and then  $n$  times gives the same result as clocking  $n$  times and then  $m$  times. For instance, clocking twice and then three times gives the same result as clocking three times and then twice. The different possible clocking operations commute. So for instance the attacker may guess that, after ten clocks of the overall generator, the 89-stage register has had two single-clocks, three double-clocks, three triple-clocks and two quadruple-clocks; she doesn't need to guess the order in which the different clockings occurred.
- (b) Furthermore, clocking once and then four times gives the same end result as clocking twice and then three times. There are lots of combinations that give, for example, 25 clocks of the register after 10 clocks of the overall generator; the attacker can assign a single overall probability to this event, without having to distinguish between the many different clocking combinations that could have led to it. This further improves the efficiency of a statistical attack.
- (c) Finally, 25 clocks of the 89-stage register may have occurred after ten generator clocks, or after nine generator clocks, or after eleven generator clocks, .... Again, this can be used to make attacks more efficient — see [5, 9] for an example.

The principles behind the design of MICKEY 2.0 are:

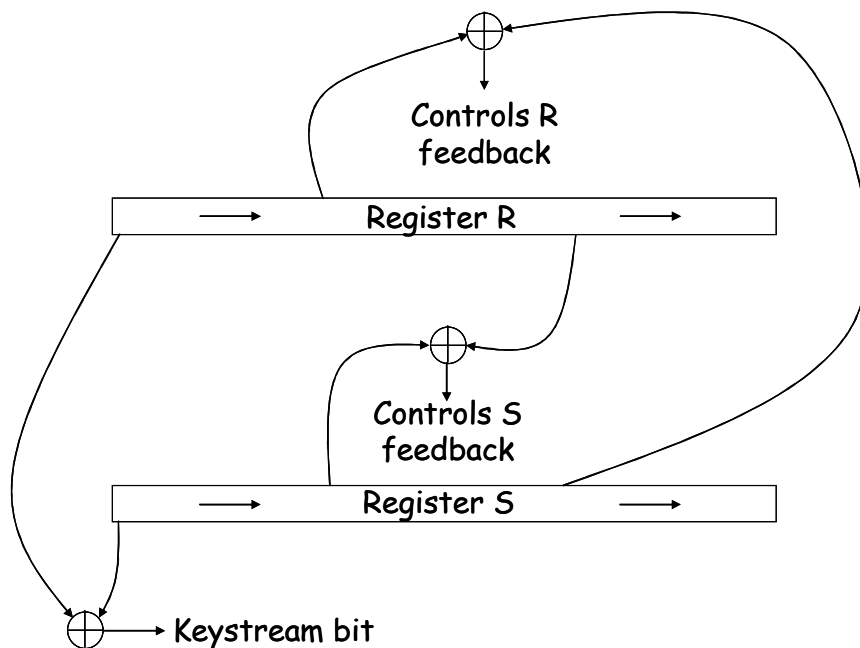
- to take all of the benefits of variable clocking, in protecting against many forms of attack;
- to guarantee period and local randomness;

- subject to those, to reduce the susceptibility to statistical attacks as far as possible.

Specifically, taking points (a)–(c) in turn:

- (a) does apply to register  $\mathcal{R}$  (because  $\text{clock}^{\mathcal{J}} \circ \text{clock}^1 = \text{clock}^1 \circ \text{clock}^{\mathcal{J}}$ ), but does not apply to register  $\mathcal{S}$ , whose different clocking operations do not commute.
- (b) does not apply to either register. In the case of  $\mathcal{R}$ , for any given values  $t \leq 2^{40}$  and  $u$ , there is at most one possible pair of values  $n_1$  and  $n_{\mathcal{J}}$  such that  $0 \leq n_1, n_{\mathcal{J}} \leq t$ ;  $n_1 + n_{\mathcal{J}} = t$ ; and  $n_1 + n_{\mathcal{J}}\mathcal{J} = u$ . ( $n_1$  and  $n_{\mathcal{J}}$  represent the number of times that  $\mathcal{R}$  is clocked once and  $\mathcal{J}$  times respectively.)
- (c) does not apply to either register. In the case of  $\mathcal{R}$ , for any given value  $u$ , there is at most one possible triple of values  $t$ ,  $n_1$  and  $n_{\mathcal{J}}$  such that  $t \leq 2^{40}$ ;  $0 \leq n_1, n_{\mathcal{J}} \leq t$ ;  $n_1 + n_{\mathcal{J}} = t$ ; and  $n_1 + n_{\mathcal{J}}\mathcal{J} = u$ .

In MICKEY 2.0, the register  $\mathcal{R}$  acts as the “engine”, ensuring that the state of the generator does not repeat within the generation of a single keystream sequence, and ensuring good local statistical properties. The influence of  $\mathcal{R}$  on the clocking of  $\mathcal{S}$  also prevents  $\mathcal{S}$  from becoming stuck in a short cycle. If the “jump index”  $\mathcal{J} < 2^{60}$ , then the state of  $\mathcal{R}$  will not repeat during the generation of a maximum length ( $2^{40}$ -bit) keystream sequence; and if  $\mathcal{J} > 2^{40}$ , then property (c) above is satisfied. We chose the “jump index”  $\mathcal{J}$  as to have the largest possible value subject to  $\mathcal{J} < 2^{50}$ .



**Figure 3:** The variable clocking architecture

### 7.3 Selection of clock control bits

We deliberately chose the clock control bits for each register to be derived from both registers, in such a way that knowledge of either register state is not sufficient to tell the attacker how either register will subsequently be clocked. This helps to guard against “guess and determine” or “divide and conquer” attacks.

### 7.4 The $S$ register feedback function

For any fixed value of  $CONTROL\_BIT\_S$ , the clocking function of  $S$  is invertible (so that the space of possible register values is not reduced by clocking  $S$ ).

Our design goal for the clocking function of  $S$  can be stated as follows. Assume that the initial state of  $S$  is randomly selected, and that the sequence of values of  $CONTROL\_BIT\_S$  applied to the clocking of  $S$  are also randomly selected. Then consider the sequence  $(s_0(i) : i = 0, 1, 2, \dots)$ . (By  $s_0(i)$  we mean the contents of  $s_0$  after the generator has been clocked  $i$  times.) We want to avoid any strong affine relations in that sequence — that is, we do not want there to exist a set  $I$  such that the value  $p = \sum_{i \in I} s_0(i)$  is especially likely to be equal to 0 (or to 1) as the initial state and  $CONTROL\_BIT\_S$  range over all possible values.

The reason for this design goal is to avoid attacks based on establishing a probabilistic linear model (i.e. a set  $I$  as described above) that would allow a linear combination of keystream bits to be strongly correlated to a combination of bits only from the (“linear”, “weaker”)  $R$  register. We are thinking here especially of distinguishing attacks.

It is not straightforward to meet this design goal in an optimum sense (even if we defined it more precisely than we have done), but we do have some reason to believe that we have met it pretty well. At least, earlier proposals we considered for  $S$  were weaker in this regard. We modelled a number of constructions on a scaled down version of  $S$ , and looked for the strongest linear relations holding over relatively short sequences  $(s_0(i))$ , and we found that the construction we have chosen performed well.

In particular, our construction preserves local randomness, in the sense that, if the initial state is uniformly random, then a sequence of 100 successive bits  $s_0(i)$  will also be uniformly random. So no sum of fewer than 101 successive bits  $s_0(i)$  will be equal to 0 with probability distinct from  $\frac{1}{2}$ . From our empirical analysis, we believe that the strongest bias will come from a combination selected from precisely 101 successive bits  $s_0(i)$ .



## 8. Changes from MICKEY version 1

In MICKEY version 1, the  $\mathcal{R}$  and  $\mathcal{S}$  registers were each 80 stages long (instead of 100). The overall state size was thus 160 bits, for an algorithm supporting an 80-bit secret key. MICKEY version 1 was, deliberately, a minimalist algorithm with very little “padding” to bolster its security margin.

The best cryptanalytic efforts against MICKEY version 1 are by Jin Hong and Woo-Hwan Kim [6]. They consider three areas of (arguable) vulnerability. The revisions in MICKEY 2.0 have been precisely targeted at addressing the issues raised in [6]. We explain the details in the following sections.

### 8.1 What are the changes from MICKEY version 1?

The changes are very simple: the two registers have each been increased from 80 stages to 100 stages. Some detailed values, such as control bit tap locations, have been scaled accordingly. There are no other changes.

### 8.2 Time-Memory-Data (TMD) tradeoff, with or without BSW sampling

Let  $\mathcal{N}$  be the size of the keystream generator state space (so  $2^{160}$  for MICKEY version 1). Let  $\mathcal{X}$  be the set of all possible keystream generator states. Let  $f : \mathcal{X} \rightarrow \mathcal{Y}$  be the function that maps a generator state to the first  $\log_2(\mathcal{N})$  bits of keystream produced. Suppose the attacker has harvested a large number of  $\log_2(\mathcal{N})$ -bit keystream sequences  $y_i \in \mathcal{Y}$ , and wants to identify a keystream generator state  $x \in \mathcal{X}$  such that  $f(x) = y_i$  for some  $i$ .

#### BS tradeoff

The Biryukov-Shamir TMD [2] algorithm succeeds with high probability if the following conditions are satisfied:

$$TM^2D^2 = \mathcal{N}^2 \quad \text{and} \quad 1 \leq D^2 \leq T$$

where  $T$  is the online time complexity,  $M$  is the memory requirement, and  $D$  is the number of keystream sequences available to the attacker. The offline time complexity is  $P = \mathcal{N}/D$ .

#### BSW sampling

When we say that we can perform BSW sampling [3] with a sampling factor  $W$ , we mean that:

- there is a subset  $\mathcal{X}' \subset \mathcal{X}$  with cardinality  $\mathcal{N}/W$ , and it is easy to generate elements of  $\mathcal{X}'$
- if  $\mathcal{Y}'$  is the image of  $\mathcal{X}'$  under  $f$ , then it is easy to recognise elements of  $\mathcal{Y}'$ .

Our attacker may consider only those keystream sequences that are elements of  $\mathcal{Y}'$ , and apply the BS tradeoff to the problem of inverting the restricted function  $f' : \mathcal{X}' \rightarrow \mathcal{Y}'$ . If the total number of keystream sequences available to the attacker is  $D$ , only roughly  $D/W$  of these will fall in  $\mathcal{Y}'$  and so be usable; on the other hand, the size of the set of preimages is now  $\mathcal{N}/W$  instead of  $\mathcal{N}$ . The conditions for success become

$$TM^2 \left( \frac{D}{W} \right)^2 = \left( \frac{\mathcal{N}}{W} \right)^2 \quad \text{and} \quad 1 \leq \left( \frac{D}{W} \right)^2 \leq T$$

$$\text{i.e. } TM^2D^2 = \mathcal{N}^2 \quad \text{and} \quad W^2 \leq D^2 \leq TW^2$$

and the offline time complexity remains  $P = \frac{(N/W)}{(D/W)} = N/D$ . Also, very importantly, the number of table lookups in the online attack is reduced by a factor  $W$ , which greatly reduces the actual time it takes.

### TMD tradeoff against MICKEY version 1

Hong and Kim [6] show that BSW sampling can be performed on MICKEY version 1 with a sampling factor  $W = 2^{27}$ . This allows a TMD tradeoff attack to be performed with the following complexity, for instance:

- unfiltered data complexity  $D = 2^{60}$ , e.g.  $2^{20}$  keystream sequences each of length roughly  $2^{40}$  bits; filtering these by BSW sampling means that the attack is performed against a reduced set of  $D/W = 2^{33}$  keystream sequences;
- search space of reduced size  $N/W = 2^{133}$ ;
- time complexity  $T = 2^{66}$ ;
- memory complexity  $M = 2^{67}$ ;
- offline time complexity  $P = 2^{100}$ .

So we have an attack whose online time, data and memory complexities are all less than the key size of  $2^{80}$ . However, the one-off precomputation time complexity is greater than  $2^{80}$ . Other parameter values are possible, but the precomputation time is always greater than  $2^{80}$ .

There is no consensus as to whether this constitutes a successful attack. Some authors seem to ignore precomputation time completely, and consider only online complexity to matter; others would say that an attack requiring overall complexity greater than exhaustive search is of no practical significance. Although we incline more towards the second view, we recognise that some will deem the cipher less than fully secure if such attacks exist.

### MICKEY 2.0

In MICKEY 2.0, the state size  $N = 2^{200}$ . Thus, for any BS tradeoff attack, with or without BSW sampling, if  $TM^2D^2 = N^2$  then at least one of  $T$ ,  $M$  or  $D$  must be at least  $2^{80}$ . So no attack is possible with online complexity faster than exhaustive key search.

Earlier papers (e.g. [1]) have recommended that the state size of a keystream generator should be at least twice the key size, to protect against what is now usually called the Babbage-Golic TMD attack. By making the state size at least 2.5 times the key size, we also provide robust protection against the Biryukov-Shamir TMD attack, with or without BSW sampling<sup>1</sup>. This rather simple observation has not appeared in previous literature, as far as we have been able to discover.

### BSW sampling of MICKEY 2.0

It is still possible to perform BSW sampling on MICKEY 2.0. We have made no attempt to prevent this — we see no reason to do so that would justify an additional complication to the cipher design.

---

<sup>1</sup> We refer here only to TMD attacks to invert the function mapping keystream generator state to keystream. We are not talking about the function mapping key and IV to keystream, as discussed by Hong and Sarkar in [7].

### 8.3 State entropy loss and keystream convergence

The variable clocking mechanism in MICKEY means that the state entropy reduces as the generator is clocked. This is fundamental to the MICKEY design philosophy.

For MICKEY version 1, Hong and Kim [6] show that this entropy loss can result in the convergence of distinct keystream sequences within the parameters of legitimate use of the cipher. For example, if  $V$  keystream sequences of length  $2^{40}$  are generated from different  $(K, IV)$  pairs, then for large enough  $V$  there will be state collisions — and of course, once identical states are reached, subsequent keystream sequences are identical. An exact analysis seems difficult, but it appears that  $V$  may not have to be much larger than  $2^{22}$  before collisions will begin to occur.

This uncomfortable property holds because, after the generator has been run for long enough to produce a  $2^{40}$ -bit sequence, the state entropy will have reduced by nearly 40 bits, from the initial  $2^{160}$  to only just over  $2^{120}$ . Because 120 is less than twice the key size, we begin to see collisions within an amount of data less than the key size.

In MICKEY 2.0, the state size is 200 bits, and the maximum permitted length of a single keystream sequence is  $2^{40}$  bits. After the generator has been run for long enough to produce a  $2^{40}$ -bit sequence, the entropy will still be just over 160 bits. This is twice the key size, and so we no longer have a problem.

### 8.4 Weak keys

There is an obvious “lock-up” state for the register  $R$ : if the key and IV loading and initialisation leaves  $R$  in the all zeroes state, then it will remain permanently in that state. For MICKEY version 1 we reasoned as follows:

It is clear that, if an attacker assumes that this is the case, she can readily confirm her assumption and deduce the remainder of the generator state by analysing a short sequence of keystream. But, because this can be assumed to occur with probability roughly  $2^{-80}$  — much less than the probability for any guessed secret key to be correct — we do not think it necessary to prevent it (and so in the interests of efficiency we do not do so).

Hong and Kim [6] point out that there is also a lock-up state for the register  $S$ . If the key and IV loading and initialisation leaves  $S$  in this particular state, then it will remain permanently in that state, irrespective of the values of the clock control bits. The probability of a “weak state” in MICKEY version 1 is thus roughly  $2^{-79}$ . And  $2^{-79}$  is greater than  $2^{-80}$  ....

It is undoubtedly much easier to try two candidate secret keys, with a success probability of  $2^{-79}$ , than to mount an attack based on these possible weak states. So we would still argue that it is not necessary to guard against their occurrence. But anyway, with MICKEY 2.0 the increased register lengths mean that the probability of a weak state goes down to roughly  $2^{-99}$ , which is clearly too small to concern us.

## 9. The intended strength of the algorithm

When used in accordance with the rules set out in section 3, MICKEY 2.0 is intended to resist any attack faster than exhaustive key search.

The designers have not deliberately inserted any hidden weaknesses in the algorithm.

## 10. Performance of the algorithm

MICKEY 2.0 is not designed for notably high speeds in software, although it is straightforward to implement it reasonably efficiently. Our own reasonably efficient (but not turbo-charged) implementation generated  $10^8$  bits of keystream in 3.81 seconds<sup>2</sup>, using a PC with a 3.4GHz Pentium 4 processor.

There may be scope for more efficient software implementations that produce several bits of keystream at a time, making use of look-up tables to implement the register clocking and keystream derivation.

## 11. IPR

The designers of the algorithm do not claim any IPR over it, and make it freely available for any purpose. To the best of our knowledge no one else has any relevant IPR either. We will update the ECRYPT stream cipher project coordinators if we ever discover any.

## 12. References

- [1] S.Babbage, *Improved Exhaustive Search Attacks on Stream Ciphers*, European Convention on Security and Detection, IEE Conference Publication no. 408, pp 161–166, IEE, 1995.
- [2] A.Biryukov and A.Shamir, *Cryptanalytic time/memory/data tradeoffs for stream ciphers*, Asiacrypt 2000, LNCS 1976, pp1–13, Springer-Verlag, 2000.
- [3] A.Biryukov, A.Shamir and D.Wagner, *Real time cryptanalysis of A5/1 on a PC*, FSE 2000, LNCS 1978, pp1–18, Springer-Verlag, 2001.
- [4] E.Dawson, A.Clark, J.Golić, W.Millan, L.Penna, L.Simpson, *The LILI-128 Keystream Generator*, NESSIE submission, in the proceedings of the First Open NESSIE Workshop (Leuven, November 2000), and available at <http://www.cryptonessie.org>.
- [5] P.Ekdahl, T.Johansson: *Another attack on A5/1*, IEEE Transactions on Information Theory 49(1): 284-289 (2003).
- [6] J.Hong, W.Kim, *TMD-Tradeoff and State Entropy Loss Considerations of Streamcipher MICKEY*, <http://eprint.iacr.org/2005/257>. (A similar — identical? — paper is included in the proceedings of INDOCRYPT 2005.)
- [7] J.Hong, P.Sarkar, *Rediscovery of Time Memory Tradeoffs*, <http://eprint.iacr.org/2005/090>.
- [8] C.J.A.Jansen, *Streamcipher Design: Make your LFSRs jump!*, presented at the ECRYPT SASC (State of the Art in Stream Ciphers) workshop, Bruges, October 2004, and in the workshop record at <http://www.isg.rhul.ac.uk/research/projects/ecrypt/stvl/sasc-record.zip>.
- [9] A.Maximov, T.Johansson, S.Babbage, *An Improved Correlation Attack on A5/1*, in Helena Handschuh, M. Anwar Hasan (Eds.): *Selected Areas in Cryptography 2004* (ed Handschuh/Hasan), Lecture Notes in Computer Science #3357, Springer Verlag.

---

<sup>2</sup> This is faster than the figure we quoted in the MICKEY v1 specification, which may surprise the reader. We found that a slight reorganisation of our testing code allowed our compiler to make inlining optimisations that it had failed to make before. The figures we quote here are still based on the “MICKEY 2 faster” C code that we have submitted to eStream.