The stream cipher MICKEY (version 1)

Algorithm specification issue 1.0

Steve Babbage

Vodafone Group R&D, Newbury, UK
steve.babbage@vodafone.com

Matthew Dodd

Independent consultant
matthew@mdodd.net
www.mdodd.net

29th April 2005

Abstract: The stream cipher 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 (which stands for Mutual Irregular Clocking KEYstream generator).

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

2. Input and output parameters

MICKEY takes two input parameters:

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

The keystream bits output by MICKEY are labelled $z_0, z_1, \ldots$. 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 80 stages long, each stage containing one bit. We label the bits in the registers $r_0 \ldots r_{79}$ and $s_0 \ldots s_{79}$ 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:

$RTAPS = \{0,2,4,6,7,8,9,13,14,16,17,20,22,24,26,27,28,34,35,37,39,41,43,49,51,52,54,56,62,67,69,71,73,76,78,79\}$

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

- Let $r_0 \ldots r_{79}$ be the state of the register R before clocking, and let $r'_0 \ldots r'_{79}$ be the state of the register R after clocking.
- $FEEDBACK\_BIT = r_{79} \oplus INPUT\_BIT\_R$
- For $1 \le i \le 79$, $r'_i = r_{i-1}$; $r'_0 = 0$
- For $0 \le i \le 79$, if $i \in RTAPS$, $r'_i = r'_i \oplus FEEDBACK\_BIT$
- If $CONTROL\_BIT\_R = 1$:
  - For $0 \le i \le 79$, $r'_i = r'_i \oplus r_i$

4.3 Clocking the register S

Define four sequences $COMP0_1 \ldots COMP0_{78}$, $COMP1_1 \ldots COMP1_{78}$, $FB0_0 \ldots FB0_{79}$, $FB1_{0\ldots} FB1_{79}$ 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	25	26
$COMP0_i$		0	0	0	1	1	0	0	0	1	0	1	1	1	1	0	1	0	0	1	0	1	0	1	0	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	0	1
$FB0_i$	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	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	0	1

i	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
$COMP0_i$	1	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	0	0	0
$COMP1_i$	1	1	0	1	1	1	1	0	0	0	1	1	0	1	0	1	1	1	0	0	0	0	1	0	0	0	1
$FB0_i$	1	1	0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	1	0	0	1	0	1	0	1	0	0
$FB1_i$	1	0	0	1	0	1	1	0	0	0	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0

i	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	
$COMP0_i$	1	0	1	0	0	1	1	1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	0		
$COMP1_i$	0	1	1	1	0	0	0	1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	0		
$FB0_i$	1	0	1	1	1	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	1	1	0	1	0	
$FB1_i$	0	0	1	0	0	1	0	0	1	0	1	1	0	1	0	1	0	0	1	0	1	0	0	0	1	1	

We define an operation CLOCK_S (S, $INPUT\_BIT\_S$, $CONTROL\_BIT\_S$) as follows:

- Let $s_0 \ldots s_{79}$ be the state of the register S before clocking, and let $s'_0 \ldots s'_{79}$ be the state of the register after clocking. We will also use $\hat{s}_0 \ldots \hat{s}_{79}$ as intermediate variables to simplify the specification.
- $FEEDBACK\_BIT = s_{79} \oplus INPUT\_BIT\_S$
- For $1 \le i \le 78$, $\hat{s}_i = s_{i-1} \oplus ((s_i \oplus COMP0_i).(s_{i+1} \oplus COMP1_i))$; $\hat{s}_0 = 0$; $\hat{s}_{79} = s_{78}$.
- If $CONTROL\_BIT\_S = 0$:
  - For $0 \le i \le 79$, $s'_i = \hat{s}_i \oplus (FB0_i.FEEDBACK\_BIT)$
- If instead $CONTROL\_BIT\_S = 1$:
  - For $0 \le i \le 79$, $s'_i = \hat{s}_i \oplus (FB1_i.FEEDBACK\_BIT)$

4.4 Clocking the overall generator

We define an operation CLOCK_KG (R, S, $MIXING$, $INPUT\_BIT$) as follows:

- $CONTROL\_BIT\_R = s_{27} \oplus r_{53}$
- $CONTROL\_BIT\_S = s_{53} \oplus r_{26}$
- If $MIXING = TRUE$, then $INPUT\_BIT\_R = INPUT\_BIT \oplus s_{40}$; if instead $MIXING = FALSE$, then $INPUT\_BIT\_R = INPUT\_BIT$
- $INPUT\_BIT\_S = INPUT\_BIT$
- CLOCK_R (R, $INPUT\_BIT\_R$, $CONTROL\_BIT\_R$)
- CLOCK_S (S, $INPUT\_BIT\_S$, $CONTROL\_BIT\_S$)

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 $0 \le i \le IVLENGTH - 1$:
  - CLOCK_KG (R, S, $MIXING = TRUE$, $INPUT\_BIT = iv_i$)
- (Load in K.) For $0 \le i \le 79$:
  - CLOCK_KG (R, S, $MIXING = TRUE$, $INPUT\_BIT = k_i$)
- (Preclock.) For $0 \le i \le 79$:
  - CLOCK_KG (R, S, $MIXING = TRUE$, $INPUT\_BIT = 0$)

6. Generating keystream

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

- For $0 \le i \le L-1$:
  - $z_i = r_0 \oplus s_0$
  - 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 $C_R(x) = x^{80} + \sum_{i \in RTAPS} x^i$, with $INPUT\_BIT\_R$ XORed into the feedback).

If we represent elements of the field $GF(2^{80})$ as polynomials $\sum_{i=0}^{79} r_i x^i$, modulo $C_R(x)$, then shifting the register corresponds to multiplication by x in the field.

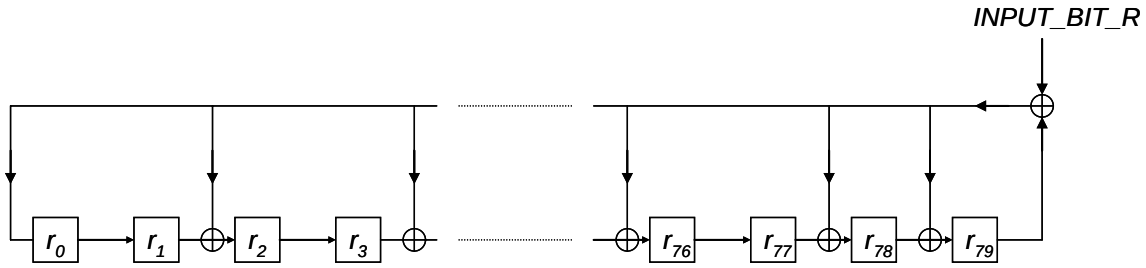

Figure 1: Clocking the R register with $CONTROL\_BIT\_R = 0$

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.

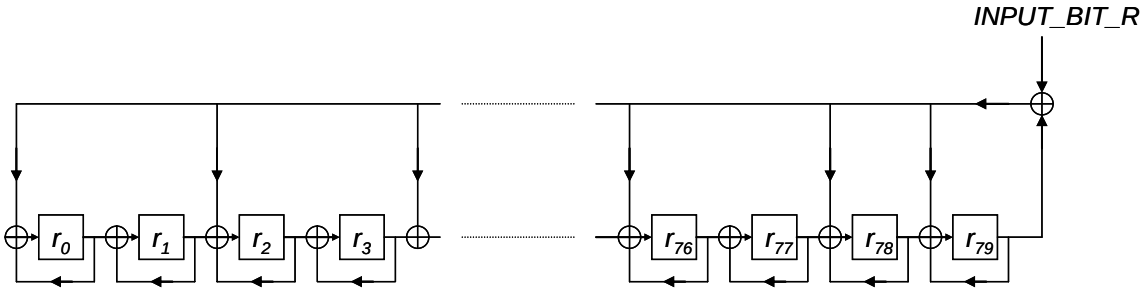

Figure 2: Clocking the R register with $CONTROL\_BIT\_R = 1$

The characteristic polynomial $C_R(x)$ has been chosen so that $C_R(x) \mid x^J + x + 1$, where $J = 2^{40} - 23$. 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 [1]. In [1], 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 [2]. 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 [3, 4] for an example.

The principles behind the design of MICKEY 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 R (because $\text{clock}^J \circ \text{clock}^1 = \text{clock}^1 \circ \text{clock}^J$), but does not apply to register S, whose different clocking operations do not commute.

(b) does not apply to either register. In the case of R, for any given values $t \le 2^{40}$ and u, there is at most one possible pair of values n_1 and n_J such that $0 \le n_1, n_J \le t$; $n_1 + n_J = t$; and $n_1 + n_J J = u$. (n_1 and n_J represent the number of times that R is clocked once and J times respectively.)

(c) effectively does not apply to either register. In the case of R, for any given value u, if we assume that $CONTROL\_BIT\_R$ is selected at random then we are very unlikely in

practice to observe more than one triple of values t, n_1 and n_J such that $t \le 2^{40}$; $0 \le n_1, n_J \le t$; $n_1 + n_J = t$; and $n_1 + n_J J = u$. (For instance, it is very unlikely that in a keystream sequence of length 2^{40} we will see $n_1 \ge J$.)

In MICKEY, the register 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 R on the clocking of S also prevents S from becoming stuck in a short cycle. We chose the "jump index" J as close as possible to 2^{40}, so that the state of R will not repeat during the generation of a maximum length (2^{40}-bit) keystream sequence, but so that property (c) above is satisfied as perfectly as possible.

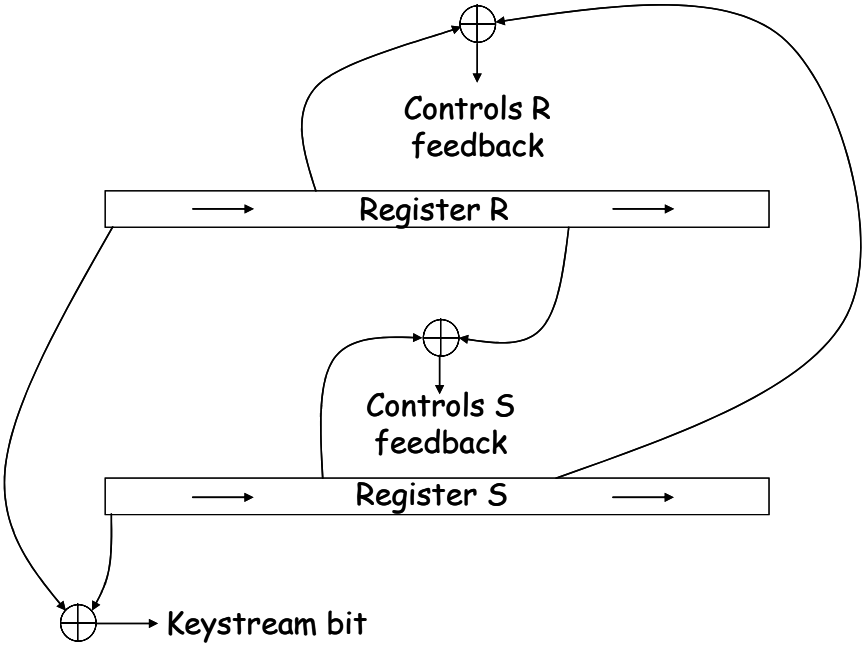

Figure 3: The variable clocking architecture

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 $(s_0(i) : i = 0,1,2,...)$. (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 80 successive bits $s_0(i)$ will also be uniformly random. So no sum of fewer than 81 successive bits $s_0(i)$ will be equal to 0 with probability distinct from ½. From our empirical analysis, we believe that the strongest bias will come from a combination selected from precisely 81 successive bits $s_0(i)$.

We should be honest, though, and say that we would ideally have liked more time to analyse possible constructions. There is probably some scope for further improvement.

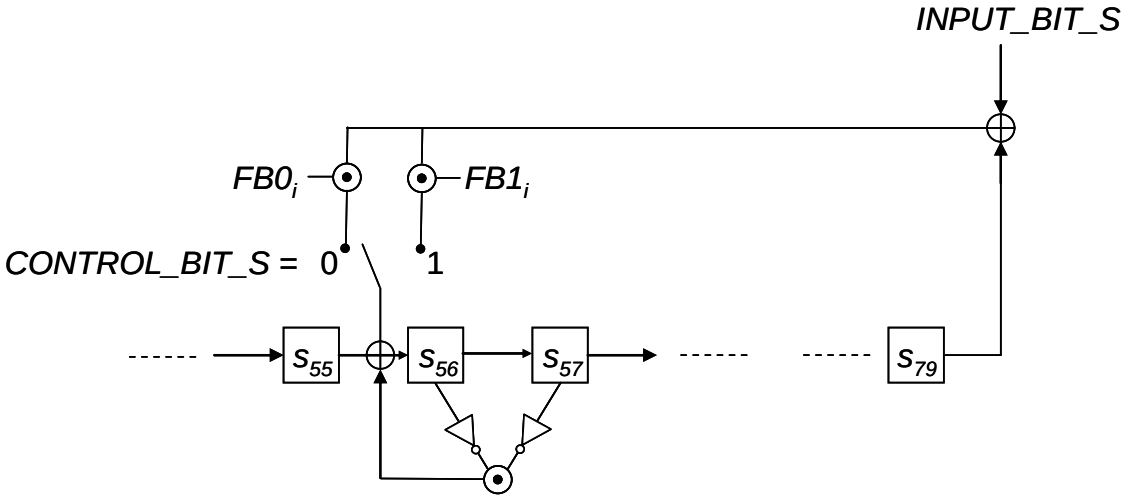

Figure 4: Clocking the S register

7.5 Key loading

We use a non-linear loading mechanism to protect against resynchronisation attacks.

7.6 Algebraic attacks

Algebraic attacks usually become possible when the keystream is correlated to one or more linearly clocking registers, whose clocking is either entirely predictable or can be guessed.

We have taken care that the attacker cannot eliminate the uncertainty about the clocking of either register by guessing a small set of values. (By illustrative contrast, some attacks on LILI-128 [2] were possible because the state of the 39-stage register could be guessed, and then the clocking of the 89-stage register became known.)

Furthermore, each keystream bit produced by MICKEY is not correlated to the contents of either one register (so in particular not to the "linear register" R).

7.7 Weak keys

There is a small class of arguably weak keys for MICKEY: namely, those (K, IV) pairs for which the state of R after loading is all zeroes. 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} — the same probability as 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).

7.8 State entropy

The generator is subject to variable clocking under control of bits from within the generator. This results in a reduction of the entropy of the overall generator state: some generator states after clocking have two or more possible preimages, and some states have no possible preimages. We considered the possibility of attacks resulting from this, but we do not believe that any exist. The fact that the control bit for each register is derived by XORing bits from both registers, and hence is uncorrelated to the state of the register it controls, is crucial: it means that clocking the overall generator does not reduce the entropy of either one register state.

7.9 Output function

MICKEY uses a very simple output function ($r_0 \oplus s_0$) to compute keystream bits from the register states.

We considered more complex alternatives, e.g. of the form $r_0 \oplus g(r_1 \ldots r_{79}) \oplus s_0 \oplus h(s_1 \ldots s_{79})$ for some Boolean functions g and h. Although these might increase the security margin against some types of attack, we preferred to keep the output function simple and elegant, and rely instead on the mutual irregular clocking of the registers.

8. The intended strength of the algorithm

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

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

9. Performance of the algorithm

MICKEY 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 5.2 seconds, 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.

10. 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.

11. References

[1] 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.

[2] 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.

[3] Patrik Ekdahl, Thomas Johansson: *Another attack on A5/1*, IEEE Transactions on Information Theory 49(1): 284-289 (2003).

[4] 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.