

The stream cipher MICKEY-128 2.0

Steve Babbage

Matthew Dodd

Vodafone Group R&D, Newbury, UK

steve.babbage@vodafone.com

Independent consultant

matthew@mdodd.net

www.mdodd.net

17th August 2006

Abstract: We present a strengthened version 2.0 of the stream cipher MICKEY-128. MICKEY-128 (which stands for Mutual Irregular Clocking KEYstream generator with a 128-bit key) is aimed at resource-constrained hardware platforms, but where a key size of 128 bits is required. 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, MICKEY-128, stream cipher, ECRYPT, irregular clocking.

This version 1.1 of the specification corrects some typos, and makes some typesetting easier to read. There are no changes to the cipher. Thanks to Joachim Strömbergson for pointing out the problems in the first version.

1. Introduction

We present the stream cipher MICKEY-128 2.0 (which stands for Mutual Irregular Clocking KEYstream generator with a 128-bit key).

MICKEY-128 2.0 is aimed at resource-constrained hardware platforms, but where a key size of 128 bits is required. It is intended to have low complexity in hardware, while providing a high level of security.

2. Input and output parameters

MICKEY-128 2.0 takes two input parameters:

- a 128-bit secret key K , whose bits are labelled $k_0 \dots k_{127}$;
- an initialisation variable IV , anywhere between 0 and 128 bits in length, whose bits are labelled $iv_0 \dots iv_{IVLENGTH-1}$.

The keystream bits output by MICKEY-128 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^{64} bits. It is acceptable to generate 2^{64} such sequences (time permitting!), 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 160 stages long, each stage containing one bit. We label the bits in the registers $r_0 \dots r_{159}$ and $s_0 \dots s_{159}$ 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 :

$$RTAPS = \{0,4,5,8,10,11,14,16,20,25,30,32,35,36,38,42,43,46,50,51,53,54,55,56,57,60,61,62,63,65,66,69,73,74,76,79,80,81,82,85,86,90,91,92,95,97,100,101,105,106,107,108,109,111,112,113,115,116,117,127,128,129,130,131,133,135,136,137,140,142,145,148,150,152,153,154,156,157\}$$

We define an operation $CLOCK_R(R, INPUT_BIT_R, CONTROL_BIT_R)$ as follows:

- Let $r_0 \dots r_{159}$ be the state of the register R before clocking, and let $r'_0 \dots r'_{159}$ be the state of the register R after clocking.
- $FEEDBACK_BIT = r_{159} \oplus INPUT_BIT_R$
- For $1 \leq i \leq 159$, $r'_i = r_{i-1}$; $r'_0 = 0$
- For $0 \leq i \leq 159$, if $i \in RTAPS$, $r'_i = r'_i \oplus FEEDBACK_BIT$
- If $CONTROL_BIT_R = 1$:
 - For $0 \leq i \leq 159$, $r'_i = r'_i \oplus r_i$

4.3 Clocking the register S

Define four sequences $COMPO_1 \dots COMPO_{158}$, $COMP1_1 \dots COMP1_{158}$, $FBO_0 \dots FBO_{159}$, $FB1_0 \dots FB1_{159}$ 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	
$COMPO_i$	1	1	1	1	0	1	0	0	1	0	0	1	1	1	1	0	1	1	0	1	0	1	1	1	0	1	0	1
$COMP1_i$	0	0	0	1	1	0	0	1	1	1	1	1	0	0	0	1	0	0	1	1	0	0	0	1	0	1	0	1
FBO_i	1	1	1	1	0	1	0	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1
$FB1_i$	1	1	0	1	0	1	0	1	1	1	1	0	1	1	1	0	0	0	1	0	1	1	1	1	1	1	1	0
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	
$COMPO_i$	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	0	1	1	0	0	1	
$COMP1_i$	1	1	1	1	0	0	0	0	1	1	0	0	1	0	0	1	1	1	1	0	0	0	1	1	0	1	1	
FBO_i	0	0	0	1	1	0	1	0	0	0	1	0	0	1	1	0	0	0	1	0	1	1	1	1	1	1	0	1
$FB1_i$	1	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	1	1	0	0	1	1	1	

<i>i</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	80
<i>COMPO_i</i>	0	0	1	0	0	1	1	1	1	0	0	1	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0
<i>COMP1_i</i>	0	1	0	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	0
<i>FBO_i</i>	0	0	0	1	1	1	0	0	0	0	1	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1	0
<i>FB1_i</i>	1	0	0	0	0	0	1	1	1	0	0	1	1	0	1	1	0	1	0	0	0	1	1	0	0	0	0
<i>i</i>	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107
<i>COMPO_i</i>	0	0	0	0	0	0	0	1	0	0	1	1	1	1	0	1	0	0	0	1	1	0	0	1	0	0	1
<i>COMP1_i</i>	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0	0	1	0	1	1	0	0
<i>FBO_i</i>	0	1	0	0	1	1	1	0	1	1	0	0	1	1	0	1	0	0	0	1	0	0	1	1	1	0	1
<i>FB1_i</i>	1	0	1	1	0	0	1	1	1	1	1	0	1	1	0	1	1	1	0	0	1	1	1	0	1	1	1
<i>i</i>	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134
<i>COMPO_i</i>	1	0	1	1	1	1	1	1	0	1	0	1	1	1	1	0	1	1	0	0	0	1	1	1	1	1	0
<i>COMP1_i</i>	0	1	1	1	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	1	0	1	0	1	1	0
<i>FBO_i</i>	0	0	1	0	0	0	1	0	1	0	1	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1
<i>FB1_i</i>	1	1	1	0	1	1	0	1	0	0	1	0	0	0	1	1	0	1	1	0	1	1	1	1	0	1	1
<i>i</i>	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159		
<i>COMPO_i</i>	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	0	0	0			
<i>COMP1_i</i>	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1				
<i>FBO_i</i>	1	1	1	0	1	0	0	0	0	1	1	0	0	0	1	1	0	1	1	0	0	0	0	0	1		
<i>FB1_i</i>	1	0	0	0	0	0	0	0	1	1	1	1	0	0	1	0	1	1	0	0	0	1	0	0	0		

We define an operation $CLOCK_S(S, INPUT_BIT_S, CONTROL_BIT_S)$ as follows:

- Let $s_0 \dots s_{159}$ be the state of the register S before clocking, and let $s'_0 \dots s'_{159}$ be the state of the register after clocking. We will also use $\hat{s}_0 \dots \hat{s}_{159}$ as intermediate variables to simplify the specification.
- $FEEDBACK_BIT = s_{159} \oplus INPUT_BIT_S$
- For $1 \leq i \leq 158$, $\hat{s}_i = s_{i-1} \oplus ((s_i \oplus COMPO_i) \cdot (s_{i+1} \oplus COMP1_i))$; $\hat{s}_0 = 0$; $\hat{s}_{159} = s_{158}$.
- If $CONTROL_BIT_S = 0$:
 - For $0 \leq i \leq 159$, $s'_i = \hat{s}_i \oplus (FBO_i \cdot FEEDBACK_BIT)$
- If instead $CONTROL_BIT_S = 1$:
 - For $0 \leq i \leq 159$, $s'_i = \hat{s}_i \oplus (FB1_i \cdot FEEDBACK_BIT)$

4.4 Clocking the overall generator

We define an operation $CLOCK_KG(R, S, MIXING, INPUT_BIT)$ as follows:

- $CONTROL_BIT_R = s_{54} \oplus r_{106}$
- $CONTROL_BIT_S = s_{106} \oplus r_{53}$

- If $MIXING = TRUE$,
 - $CLOCK_R(R, INPUT_BIT_R = INPUT_BIT \oplus s_{80}, CONTROL_BIT_R = CONTROL_BIT)$
 - $CLOCK_S(S, INPUT_BIT_S = INPUT_BIT, CONTROL_BIT_S = CONTROL_BIT)$
- If instead $MIXING = FALSE$,
 - $CLOCK_R(R, INPUT_BIT_R = INPUT_BIT, CONTROL_BIT_R = CONTROL_BIT)$
 - $CLOCK_S(S, INPUT_BIT_S = INPUT_BIT, CONTROL_BIT_S = CONTROL_BIT)$

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 $0 \leq i \leq 127$:
 - $CLOCK_KG(R, S, MIXING = TRUE, INPUT_BIT = k_i)$
- (Preclock.) For $0 \leq i \leq 159$:
 - $CLOCK_KG(R, S, MIXING = TRUE, INPUT_BIT = 0)$

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

The design principles of MICKEY-128 2.0 are exactly the same as those of MICKEY 2.0 [1]. We will not repeat them here. We have treated MICKEY-128 2.0 as a separate algorithm purely to keep the specification of each version simpler.

In section 7.1 of the MICKEY 2.0 specification [1], we mention a value $J = 2^{50} - 157$ related to the clocking of register R . For MICKEY-128 2.0, the corresponding value of J is $2^{80} - 255$.

8. Changes from MICKEY-128 version 1

The changes are very simple: the two registers have each been increased from 128 stages to 160 stages. Some detailed values, such as control bit tap locations, have been scaled accordingly. There are no other changes.

For an explanation of the rationale behind these changes, see section 8 of [1].

9. The intended strength of the algorithm

When used in accordance with the rules set out in section 3, MICKEY-128 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-128 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 4.81 seconds¹, 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.H.Babbage, M.W.Dodd, *The stream cipher MICKEY 2.0*, revised ECRYPT stream cipher submission, expected to become available via the ECRYPT web site.

¹ This is faster than the figure we quoted in the MICKEY-128 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-128 2 faster” C code that we have submitted to eStream.