The MICKEY stream ciphers

Steve Babbage[1] and Matthew Dodd[2]

[1] Vodafone Group R&D, Newbury, UK steve.babbage@vodafone.com
[2] Independent consultant matthew@mdodd.net

Abstract. The family of stream ciphers 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.

1 Introduction and overview

The MICKEY family of algorithms was designed in response to the ECRYPT 'Call for Stream Cipher Primitives' in 2005, and directed at 'Profile 2' — stream ciphers intended for use on resource-constrained hardware platforms. Specifically, it is intended to have low complexity in hardware, while providing a high level of security. In fact, two variants of the algorithm have been defined: MICKEY, with an 80-bit key, and MICKEY-128, with a 128-bit key.

'MICKEY' is an abbreviation of 'Mutual Irregular Clocking KEYstream generator', and this encapsulates the original design concept, illustrated in Figure 1. The algorithm is based around two registers R and S, each of which has two modes of clocking selected by a control bit. With this as a starting point, we were lead to design a clocking rule for the ensemble (R, S), in which the control bit for each register is formed from combination of bits dependent on both registers.

It was also intended from the outset that R should clock as a Galois-stepping feedback shift register either 1 or J times, given that J steps can be implemented efficiently in a single clock cycle by taking advantage of an idea introduced by Jansen [14]. This is discussed in detail in section 2.1 below.

The register S, on the other hand, was intended to clock non-linearly, in two different ways. Successive bits of keystream are formed by combining bits from the registers R and S. Broadly speaking, the idea was that the linearity of R would ensure good statistical properties and guarantees about period, whilst the non-linearity of S would protect against attacks that might be mounted against a linear system.

2 Design principles

In this section we describe the design, and the choices behind it, in further detail. Note that complete formal specifications of MICKEY and MICKEY-128 are provided in appendices A and B.

This section applies equally to both variants of the cipher, and we introduce the parameter n so that we can discuss both at the same time; $n = 100$ for MICKEY and $n = 160$ for MICKEY-128. Thus n is the length of register R, and, equally, the length of register S. As stipulated in sections A.2 and B.2, keystream sequences are limited to $2^{|K|/2}$ bits, and at most $2^{|K|/2}$ sequences may be produced from different IV values with a single key; here $|K|$ denotes the key length.

When used in accordance with the rules set out in sections A.2 and B.2, both MICKEY variants are intended to resist any attack faster than exhaustive key search. The designers have not deliberately inserted any hidden weaknesses in the algorithms.

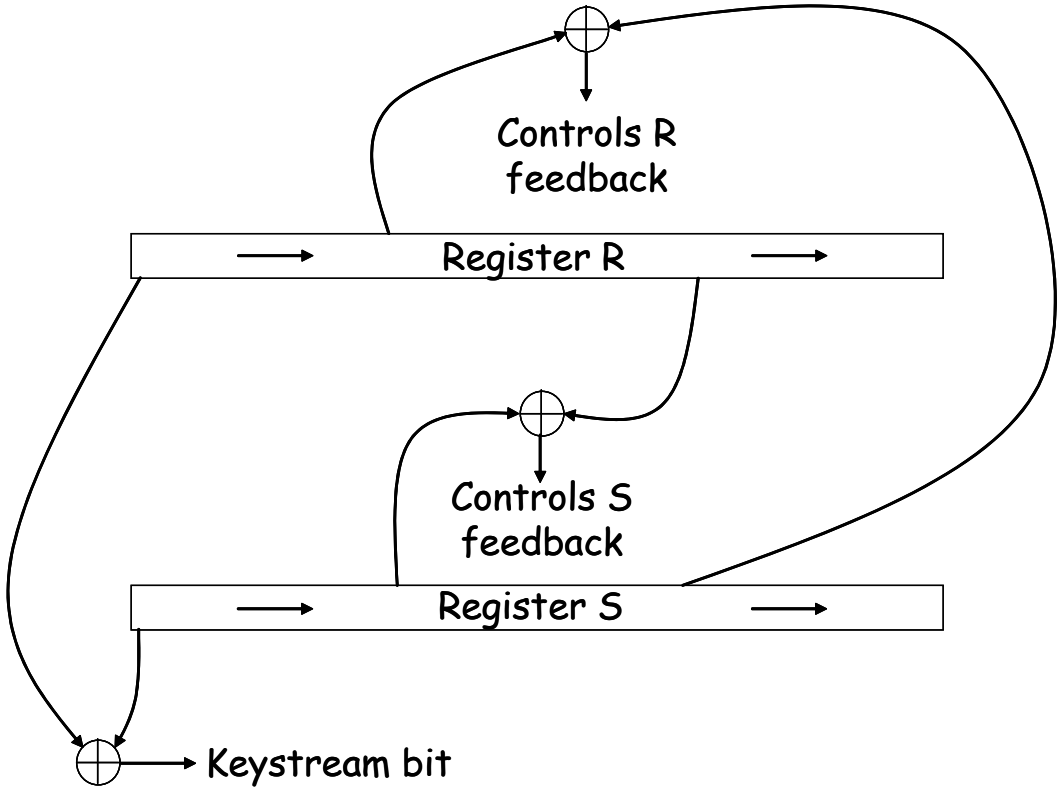

Fig. 1. MICKEY algorithm structure

The designers of MICKEY family of algorithms 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.

2.1 The variable clocking of *R*: what it does

Register R has a set of feedback taps $RTAPS$, and clocks in one of two ways according to the value of a control bit $CONTROL\_BIT\_R$. When the value of $CONTROL\_BIT\_R = 0$, the clocking of R is a standard linear feedback shift register clocking operation (with Galois-style feedback, according to the primitive characteristic polynomial $C_R(x) = x^n + \sum_{i \in RTAPS} x^i$, with $INPUT\_BIT\_R$ XORed into the feedback). This is shown in Figure 2 for the case $n = 100$.

If we represent elements of the field $GF(2^n)$ as polynomials $\sum_{i=0}^{n-1} r_i x^i$, modulo $C_R(x)$, then shifting the register corresponds to multiplication by x in the field.

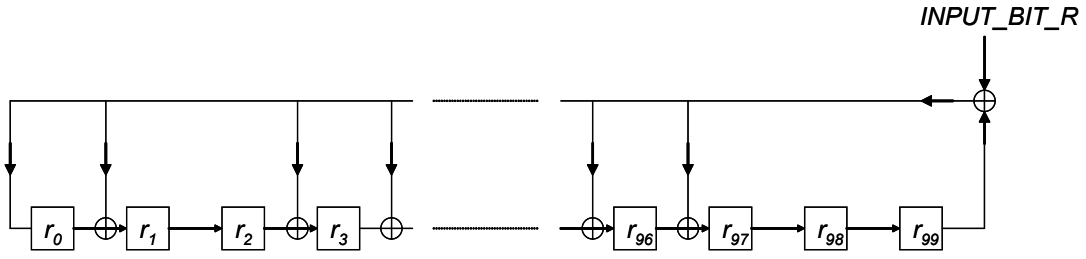

Fig. 2. Clocking the R register with $CONTROL\_BIT\_R = 0$

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 3. This corresponds to multiplication by $x + 1$ in the same field.

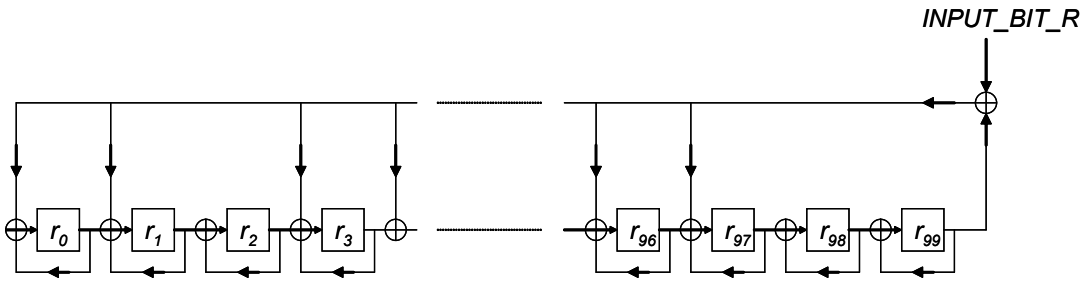

Fig. 3. Clocking the R register with $CONTROL\_BIT\_R = 1$

The characteristic polynomial $C_R(x)$ has been chosen so that $C_R(x) \mid x^J + x + 1$ where $J = 2^{50} - 157$ for MICKEY and $J = 2^{80} - 255$ for MICKEY-128 . 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 [14]. In [14], 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.

This observation is elaborated in [6], where we describe a technique, reproduced below, for finding suitable characteristic polynomials. Suppose first that $C(x)$ is a polynomial of even degree n over GF(2) that divides $x^J + x + 1$, where $J = 2^{n/2} - \delta$ for a small positive integer δ. Since $C(x) \mid x^J + x + 1$,

$$C(x) \mid x^{2^{n/2}} + x^{\delta+1} + x^{\delta}$$

Hence, if we denote congruence mod $C(x)$ by $\equiv$,

$$\begin{aligned} x^{2^n} + x &= \left(x^{2^{n/2}}\right)^{2^{n/2}} + x \\ &\equiv \left(x^{\delta+1} + x^{\delta}\right)^{2^{n/2}} + x \\ &= \left(x^{2^{n/2}}\right)^{\delta+1} + \left(x^{2^{n/2}}\right)^{\delta} + x \\ &\equiv \left(x^{\delta+1} + x^{\delta}\right)^{\delta+1} + \left(x^{\delta+1} + x^{\delta}\right)^{\delta} + x \end{aligned}$$

If moreover $C(x)$ is primitive, then $x^{2^n} + x \equiv 0$, so the polynomial

$$G_\delta(x) = \left(x^{\delta+1} + x^{\delta}\right)^{\delta+1} + \left(x^{\delta+1} + x^{\delta}\right)^{\delta} + x$$

of degree $(\delta + 1)^2$ must have $C(x)$ as a factor.

To find suitable characteristic polynomials for the MICKEY family of algorithms, we can therefore apply the following algorithm, starting at $\delta = \lceil \sqrt{n} \rceil - 1$:

- Construct $G_\delta(x)$, and see if it has any factor $F(x)$ of degree n
- If it does, check whether $F(x)$ is primitive
- If it is, then check whether $F(x)$ really does divide $x^{2^{n/2}-\delta} + x + 1$

- If it does, set $C(x) = F(x)$ and stop
- Otherwise, increment δ and start again

The following variant may be slightly more efficient:

- Compute $\gcd(G_\delta(x), x^{2^{n/2}-\delta} + x + 1)$ and factorise it
- If there is any factor $F(x)$ of degree n, check whether $F(x)$ is primitive
- If a primitive factor $F(x)$ is found, set $C(x) = F(x)$ and stop
- Otherwise, increment δ and start again

Notice, from the considerations above, that any factor of $\gcd(G_\delta(x), x^{2^{n/2}-\delta} + x + 1)$ is also a factor of $x^{2^n} + x$.

2.2 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 [9]. 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:

1. Clocking the 89-stage register r times and then s times gives the same result as clocking s times and then r 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.
2. 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.
3. 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 [10, 15] for an example.

The principles behind the design of the MICKEY algorithms 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 1 to 3 in turn:

1. 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.

2. does not apply to either register. In the case of R, for any given values $t \leq 2^{|K|/2}$ and u, there is at most one possible pair of values n_1 and n_J such that $0 \leq n_1, n_J \leq t$; $n_1 + n_J = t$; 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.)
3. does not apply to either register. In the case of R, since $J > 2^{|K|/2}$ (for either MICKEY variant), it is true that for any given value u, there is as most one triple of values t, n_1 and n_J such that $t \leq 2^{|K|/2}$; $0 \leq n_1, n_J \leq t$; $n_1 + n_J = t$; and $n_1 + n_J J = u$.

In the MICKEY family of stream ciphers, 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. If the 'jump index' $J < 2^{n-|K|/2}$, then the state of R will not repeat during the generation of a maximum length $\left(2^{|K|/2}\right)$-bit keystream sequence; and if $J > 2^{|K|/2}$, then property 3 above is satisfied. We chose the 'jump index' J to have the largest possible value subject to $J < 2^{n/2}$; then indeed both $J < 2^{n-|K|/2}$ and $J > 2^{|K|/2}$.

2.3 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.

2.4 The *S* register feedback function

The clocking rule for register S is specified in sections A.3 and B.3. Figure 4 illustrates the principle by showing the updating of the particular cell s_{56} in MICKEY. In general, the new value of a cell s_i is formed from the exclusive-or of the following:

- s_{i-1}, if $1 \leq i \leq n-1$;
- the product of $s_i \oplus COMP0_i$ and $s_{i+1} \oplus COMP1_i$, if $1 \leq i \leq n-2$, for predefined bit values $COMP0_i$ and $COMP1_i$;
- $s_{n-1} \oplus INPUT\_BIT\_S$, for certain predefined values of i which depend also on the value of the clock control bit.

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 $(s_0(i))_{i=0,1,2,\ldots}$. (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.

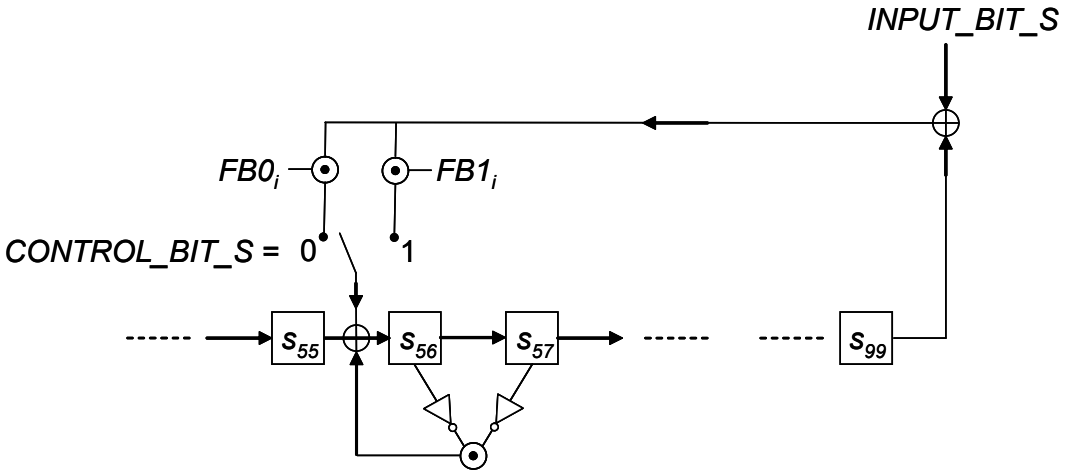

Fig. 4. Clocking the S register

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

2.5 Key loading

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

2.6 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 [9] 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).

2.7 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.

3 Register sizes

In this section we consider the choice of the parameter n.

Initially, n was chosen to be the same as the key length, and this choice was retained in the first, version 1, proposals for MICKEY and MICKEY-128 to ECRYPT [2, 3]. Subsequently this decision was revised [4, 5] in the current (version 2.0) MICKEY algorithms, so that n became 1.25 times the key length.

This change was made in response to the work of Jin Hong and Woo-Hwan Kim [12]. They considered three areas of (arguable) vulnerability, which are all addressed by this new choice for the parameter n. We explain the details in the following sections.

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

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

BS tradeoff The Biryukov-Shamir TMD [7] algorithm succeeds with high probability if the following conditions are satisfied:

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

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 $P = N/D$.

BSW sampling When we say that we can perform BSW sampling [8] with a sampling factor W, we mean that:

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

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

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

i.e.

$$TM^2D^2 = N^2 \quad \text{and} \quad W^2 \le D^2 \le TW^2$$

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 [12] 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-Golić 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[3]. 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.

3.2 State entropy loss and keystream covergence

It is fundamental to the design of the MICKEY algorithm family that the keystream 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. 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.

However, for MICKEY version 1, Hong and Kim [12] show that the overall entropy loss can result in the convergence of distinct keystream sequences within the parameters of legitimate use

[3] 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 [13]

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.

3.3 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} — the same 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 [12] point out that, for MICKEY version 1, 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.

4 Performance of the algorithm

The MICKEY cipher family 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) implementations generated 10^8 bits of keystream in 3.81 seconds for MICKEY, and in 4.81 seconds for MICKEY-128, 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.

Further information on the performance of MICKEY and MICKEY-128 in software — on various platforms — and hardware can be found via [11].

5 Afterthoughts

So how is MICKEY looking now, compared to the other eSTREAM candidates?

5.1 Security against classical cryptanalysis

In terms of security against classical cryptanalysis, we believe that MICKEY is standing up very well. The observations of Hong and Kim [12] on the MICKEY version 1 ciphers are all fully addressed in the current versions. No other threatening analysis has emerged, despite the efforts of some very good cryptanalysts.

5.2 Security against side channel attacks

If security against side channel attacks is required, then MICKEY is perhaps not optimal. The main area of susceptibility is the variable clocking of the linear register R. When $CONTROL\_BIT\_R = 1$, the additional XORs will consume more power in a naïve implementation.

By contrast, the eSTREAM submission Pomaranch also uses the "jumping" idea, but in such a way that half of the cells in a register have an XOR when the control bit takes one value, and the other half do when the control bit takes the other value. So the overall power consumption is likely to be the same. A similar approach could have been taken with MICKEY, and would give readier protection against power analysis attacks.

Having said that, we think that side channel attacks are largely irrelevant in the great majority of real world stream cipher applications. The legitimate user of an encrypting device has no motivation to extract their own encryption key (whereas they may, for instance, be motivated to clone their own SIM card or Pay-TV card). And if an outsider has close enough access to the encrypting device to carry out attacks of this kind, then there are more obvious bad things that she can do. It is possible to think up use cases in which side channel attacks on a stream cipher might matter, but they are not typical.

5.3 Performance

MICKEY's main performance goal is to run at very low power, or with very few logic gates, in resource-constrained hardware. As such, it compares very well with other eSTREAM submissions; it is indeed one of the very smallest.

Some other submissions have been designed to allow faster operation than MICKEY, by allowing a much greater degree of pipelining. Trivium is the most extreme example. The variable clocking approach taken in MICKEY does not lend itself well to pipelining.

So overall we think that MICKEY is a good choice where power or gate count are the prime performance considerations; less so where the highest speeds are required.

5.4 Conclusion

The evidence so far from the eSTREAM process is that MICKEY is a high security cipher, well suited to stream cipher applications where very low power or gate count are required.

A Specification of the cipher MICKEY

In this appendix, we provide a full specification of the stream cipher MICKEY (version 2.0).

A.1 Input and output parameters

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

A.2 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.

A.3 Components of the keystream generator

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'.

Clocking the register R Define a set of feedback tap positions for R:

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

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 $1 \le i \le 99$, $r'_i = r_{i-1}$; $r'_0 = 0$
– For $0 \le i \le 99$, if $i \in RTAPS$, $r'_i = r'_i \oplus FEEDBACK\_BIT$
– If $CONTROL\_BIT\_R = 1$:
  • For $0 \le i \le 99$, $r'_i = r'_i \oplus r_i$

Clocking the register S Define four sequences $(COMP0_i)_{i=1}^{98}$, $(COMP1_i)_{i=1}^{98}$, $(FB0_i)_{i=0}^{99}$ and $(FB1_i)_{i=0}^{99}$ according to Table 1.

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

– Let $s_0 \dots s_{99}$ be the state of the register S before clocking, and $s'_0 \dots s'_{99}$ 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.
– $FEEDBACK\_BIT = s_{99} \oplus INPUT\_BIT\_S$
– For $1 \le i \le 98$, $\hat{s}_i = s_{i-1} \oplus ((s_i \oplus COMP0_i) \,.\, (s_{i+1} \oplus COMP1_i))$; $\hat{s}_0 = 0$; $\hat{s}_{99} = s_{98}$.
– If $CONTROL\_BIT\_S = 0$:
  • For $0 \le i \le 99$, $s'_i = \hat{s}_i \oplus (FB0_i \,.\, FEEDBACK\_BIT)$
– If instead $CONTROL\_BIT\_S = 1$:
  • For $0 \le i \le 99$, $s'_i = \hat{s}_i \oplus (FB1_i \,.\, FEEDBACK\_BIT)$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$COMP0_i$		0	0	0	1	1	0	0	0	1	0	1	1	1	1	0	1	0	0	1
$COMP1_i$		1	0	1	1	0	0	1	0	1	1	1	1	0	0	1	0	1	0	0
$FB0_i$	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	0	0	1	0	1
$FB1_i$	1	1	1	0	1	1	1	0	0	0	0	1	1	1	0	1	0	0	1	1

i	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
$COMP0_i$	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1	0	0	0	0
$COMP1_i$	0	1	1	0	1	0	1	1	1	0	1	1	1	1	0	0	0	1	1	0
$FB0_i$	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	1
$FB1_i$	0	0	0	1	0	0	1	1	0	0	1	0	1	1	0	0	0	1	1	0

i	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
$COMP0_i$	0	0	0	1	0	1	0	1	0	1	0	0	0	0	1	0	1	0	0	1
$COMP1_i$	1	0	1	1	1	0	0	0	0	1	0	0	0	1	0	1	1	1	0	0
$FB0_i$	1	1	0	0	1	0	0	1	0	1	0	1	0	0	1	0	1	1	1	1
$FB1_i$	0	0	0	0	1	1	0	1	1	0	0	0	1	0	0	0	1	0	0	1

i	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
$COMP0_i$	1	1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	0	1
$COMP1_i$	0	1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	0	0
$FB0_i$	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	1	1	0	1	0	1	0	0	1	0	1	0	0	0	1	1

i	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
$COMP0_i$	0	1	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	1	
$COMP1_i$	0	1	0	0	0	0	1	1	1	0	0	0	1	0	0	1	1	0	0	
$FB0_i$	0	0	1	1	0	1	1	1	0	0	1	1	1	0	0	1	1	0	0	0
$FB1_i$	1	1	0	1	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	1

Table 1. S register tables for MICKEY

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

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

A.4 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, $MIXING = TRUE$, $INPUT\_BIT = 0$)

A.5 Generating keystream

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

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

B Specification of the cipher MICKEY-128

In this appendix, we provide a full specification of the stream cipher MICKEY-128 (version 2.0).

B.1 Input and output parameters

MICKEY-128 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 are labelled $z_0, z_1, \dots$. Ciphertext is produced from plaintext by bitwise XOR with keystream bits, as in most stream ciphers.

B.2 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.

B.3 Components of the keystream generator

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'.

Clocking the register R Define a set of feedback tap positions for R:

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

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 $0 \leq i \leq 159$, $r'_i = r'_i \oplus r_i$

Clocking the register S Define four sequences $(COMP0_i)_{i=1}^{158}$, $(COMP1_i)_{i=1}^{158}$, $(FB0_i)_{i=0}^{159}$ and $(FB1_i)_{i=0}^{159}$ according to Table 2.

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

- Let $s_0 \dots s_{159}$ be the state of the register S before clocking, and $s'_0 \dots s'_{159}$ 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 $1 \leq i \leq 158$, $\hat{s}_i = s_{i-1} \oplus ((s_i \oplus COMP0_i) \,.\, (s_{i+1} \oplus COMP1_i))$; $\hat{s}_0 = 0$; $\hat{s}_{159} = s_{158}$.
- If $CONTROL\_BIT\_S = 0$:
  - For $0 \leq i \leq 159$, $s'_i = \hat{s}_i \oplus (FB0_i \,.\, FEEDBACK\_BIT)$
- If instead $CONTROL\_BIT\_S = 1$:
  - For $0 \leq i \leq 159$, $s'_i = \hat{s}_i \oplus (FB1_i \,.\, FEEDBACK\_BIT)$

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

- If $MIXING = TRUE$,
  - CLOCK_R (R, $INPUT\_BIT\_R = INPUT\_BIT \oplus s_{80}$, $CONTROL\_BIT\_R = s_{54} \oplus r_{106}$)
- If instead $MIXING = FALSE$,
  - CLOCK_R (R, $INPUT\_BIT\_R = INPUT\_BIT$, $CONTROL\_BIT\_R = s_{54} \oplus r_{106}$)
- CLOCK_S (S, $INPUT\_BIT\_S = INPUT\_BIT$, $CONTROL\_BIT\_S = s_{106} \oplus r_{53}$)

B.4 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, $MIXING = TRUE$, $INPUT\_BIT = 0$)

B.5 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, $MIXING = FALSE$, $INPUT\_BIT = 0$)

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. S. H. Babbage, M. W. Dodd, *The stream cipher MICKEY (version 1), Algorithm specification Issue 1.0*, ECRYPT stream cipher submission, in the proceedings of the SKEW Workshop (Århus, May 2005), and available at `http://www.ecrypt.eu.org/stream/ciphers/mickey/mickey.pdf`.
3. S. H. Babbage, M. W. Dodd, *The stream cipher MICKEY-128 (version 1), Algorithm specification Issue 1.0*, ECRYPT stream cipher submission, in the proceedings of the SKEW Workshop (Århus, May 2005), and available at `http://www.ecrypt.eu.org/stream/ciphers/mickey128/mickey128.pdf`.
4. S. H. Babbage, M. W. Dodd, *The stream cipher MICKEY 2.0*, revised ECRYPT stream cipher submission, `http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickeyp3.pdf`.
5. S. H. Babbage, M. W. Dodd, *The stream cipher MICKEY-128 2.0*, revised ECRYPT stream cipher submission, `http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey128_p3.pdf`.
6. S. H. Babbage, M. W. Dodd, *Finding Characteristic Polynomials with Jump Indices*, `http://eprint.iacr.org/2006/010`.
7. A. Biryukov and A. Shamir, *Cryptanalytic time/memory/data tradeoffs for stream ciphers*, Asiacrypt 2000, LNCS 1976, pp1–13, Springer-Verlag, 2000.
8. 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.
9. 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`.
10. P. Ekdahl, T. Johansson: *Another attack on A5/1*, IEEE Transactions on Information Theory 49(1): 284-289 (2003).
11. Algorithm performance pages on the eStream web site: `http://www.ecrypt.eu.org/stream/sw.html` and `http://www.ecrypt.eu.org/stream/hw.html`.
12. 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.)
13. J. Hong, P. Sarkar, *Rediscovery of Time Memory Tradeoffs*, `http://eprint.iacr.org/2005/090`.
14. 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`.
15. 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.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$COMP0_i$		1	1	1	1	0	1	0	0	1	0	0	1	1	1	1	0	1	1	0
$COMP1_i$		0	0	0	1	1	0	0	1	1	1	1	1	0	0	0	1	0	0	1
$FB0_i$	1	1	1	1	0	1	0	1	1	1	1	1	1	0	0	0	0	0	1	1
$FB1_i$	1	1	0	1	0	1	0	1	1	1	1	0	1	1	1	0	0	0	1	0

i	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
$COMP0_i$	1	0	1	1	1	0	1	1	1	0	1	0	1	0	1	0	1	0	1	0
$COMP1_i$	1	0	0	0	1	0	1	1	1	1	1	0	0	0	0	1	1	0	0	1
$FB0_i$	1	1	0	0	0	0	1	0	0	0	1	1	0	1	0	0	0	1	0	0
$FB1_i$	1	1	1	1	1	1	0	1	1	0	0	1	0	0	0	0	1	0	0	1

i	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
$COMP0_i$	1	0	0	1	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0	1
$COMP1_i$	0	0	1	1	1	1	0	0	0	1	1	0	1	1	0	1	0	1	1	1
$FB0_i$	1	1	0	0	0	1	0	1	1	1	1	1	0	1	0	0	0	1	1	1
$FB1_i$	0	0	1	1	0	0	0	1	1	0	0	1	1	1	1	0	0	0	0	0

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

i	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
$COMP0_i$	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	0	1	0	0	0
$COMP1_i$	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0
$FB0_i$	1	0	1	0	0	1	1	1	0	1	1	0	0	1	1	0	1	0	0	0
$FB1_i$	0	1	0	1	1	0	0	1	1	1	1	1	0	1	1	0	1	1	1	0

i	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
$COMP0_i$	1	1	0	0	1	0	0	1	1	0	1	1	1	1	1	1	0	1	0	1
$COMP1_i$	0	0	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	1	1	0
$FB0_i$	1	0	0	1	1	1	0	1	0	0	1	0	0	0	1	0	1	0	1	0
$FB1_i$	0	1	1	1	0	1	1	1	1	1	1	0	1	1	0	1	0	0	1	0

i	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139
$COMP0_i$	1	1	1	0	1	1	0	0	0	1	1	1	1	1	0	1	0	1	1	0
$COMP1_i$	0	1	1	0	0	1	1	0	1	0	1	0	1	1	0	1	1	1	0	1
$FB0_i$	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1	1	1	1	0	1
$FB1_i$	0	0	1	1	0	1	1	0	1	1	1	1	0	1	1	1	0	0	0	0

i	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
$COMP0_i$	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	0	0	0	
$COMP1_i$	1	0	1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	
$FB0_i$	0	0	0	0	1	1	0	0	0	1	1	0	1	1	0	0	0	0	0	1
$FB1_i$	0	0	0	1	1	1	1	0	0	1	0	1	1	0	0	0	1	0	0	0

Table 2. S register tables for MICKEY-128