DECIM, a new stream cipher for hardware applications *

C. Berbain[1], O. Billet[1], A. Canteaut[2], N. Courtois[3], B. Debraize[3,4], H. Gilbert[1], L. Goubin[4], A. Gouget[5], L. Granboulan[6], C. Lauradoux[2], M. Minier[2], T. Pornin[7] and H. Sibert[5]

Abstract

DECIM is a new stream cipher designed for hardware applications with restricted resources. The design of the cipher is based on both a nonlinear filter LFSR and an irregular decimation mechanism recently introduced and called the ABSG. Apart from the security aspects, the design goal is to produce a stream cipher with a compact hardware implementation and operating at high rates. Excluding the tricky case of Time-Memory-Data trade-off attacks, the best attacks that have been identified by the authors are at least as difficult as exhaustive search.

Keywords: synchronous stream ciphers, hardware applications, irregular decimation, nonlinear filter.

1 Introduction

DECIM is a binary additive stream cipher, that is, a synchronous stream cipher in which the keystream Z, the plaintext M and the ciphertext C are sequences of binary digits; the sequence Z is added bitwise to the plaintext M to produce the ciphertext C.

The most important aspect for the cipher is its resistance against different attacks. One of the most serious threat against the security of stream cipher systems is the class of algebraic attacks. It consists in finding a large number of algebraic relations that relate the running key and a linear combination of the secret key bits. When the cipher is clocked in a simple and/or easily predictable way, finding such algebraic relations is most of the time not difficult. One interesting method to make the relations between the running key and a linear combination

[1]France Télécom Recherche et Développement, 38/40 rue du Général Leclerc, F-92794 Issy les Moulineaux cedex 9, {come.berbain,olivier.billet,henri.gilbert}@francetelecom.com

[2]INRIA-Rocquencourt, projet CODES, domaine de Voluceau, B.P. 105, F-78153 Le Chesnay cedex, {anne.canteaut,marine.minier,cedric.lauradoux}@inria.fr

[3]Axalto Smart Cards, 36-38, rue de la Princesse - B.P. 45, F-78431 Louveciennes cedex, {ncourtois,bdebraize}@axalto.com

[4]Laboratoire PRiSM, Université de Versailles, 45 avenue des Etats-Unis, F-78035 Versailles cedex, louis.goubin@prism.uvsq.fr

[5]France Télécom Recherche et Développement, 42 rue des Coutures, BP 6243, F-14066 Caen cedex, {aline.gouget,herve.sibert}@francetelecom.com

[6]Département d'Informatique, École Normale Supérieure, 45 rue d'Ulm, F-75230 Paris cedex 05, louis.granboulan@ens.fr

[7]Cryptolog International, 16-18 rue Vulpian, F-75013 Paris, thomas.pornin@cryptolog.com

*Work partially supported by the French Ministry of Research RNRT Project "X-CRYPT" and by the European Commission via ECRYPT network of excellence IST-2002-507932.

of the secret key bits more complex is to add a component that would de-synchronize the output bits of the cipher from the clock of the LFSR.

Taking into account the last remark, DECIM has been developed around the ABSG mechanism which was first presented at the ECRYPT Workshop State of the art of stream ciphers [10] and published in [11]. The ABSG is a scheme that, like the Shrinking Generator (SG) [5] and the Self-Shrinking Generator (SSG) [17], provides a method for irregular decimation of pseudorandom sequences. The ABSG has the advantage on the one hand over the SG that it operates on a single input sequence instead of two and on the other hand over the SSG that it operates at a rate 1/3 instead of 1/4 (i.e. producing n bits of the output sequence requires on average $3n$ bits of the input sequence instead of $4n$ bits). The best known attack on the ABSG mechanism [11] when filtering a maximum-length LFSR of length L has complexity $\mathcal{O}(2^{\frac{L}{2}})$ and requires $\mathcal{O}(L2^{\frac{L}{2}})$ keystream bits.

The general running of DECIM consists in generating a pseudorandom sequence clocked in a predictable way, that is, the output of a maximum-length LFSR filtered by a symmetric Boolean function, which is next filtered by the ABSG mechanism.

The outline of the paper is as follows. In Section 2, we give an overview of DECIM. In Section 3, we provide a full description of DECIM and we explain the design choices in Section 4. In Section 5, we give an overview of different methods for cryptanalysis. In Section 6, we discuss the hardware implementation of DECIM. Finally, we sum up the strengths and advantages of DECIM in Section 7.

2 Overview of Decim

DECIM takes a 80-bit length secret key K, a plaintext message M and a 64-bit[1] length public initialization vector IV as an input to produce the ciphertext C.

The size of the inner state of DECIM is 192 bits. Both the IV injection mechanism and the output function rely on the following components:

1. a maximum-length LFSR of length 192 with a primitive feedback polynomial P;
2. a Boolean function f of 7 variables; f is used twice for filtering the internal state of the LFSR;
3. the ABSG decimation mechanism.

In addition, we describe a buffering mechanism in order to guarantee a constant throughput for the keystream.

The keystream generation mechanism is described in Figure 1. The bits of the internal state of the LFSR are numbered from 0 to 191, and they are denoted by $(x_0, \ldots, x_{191})$. The Boolean function f is involved twice in the keystream generation; we denote by f_1 and f_2 the two uses of the Boolean function.

The sequence output by the LFSR, that is, the sequence of the linear feedback values, is denoted by $\mathbf{s} = (s_t)_{t\geq 0}$. The sequences output by the functions f_1 and f_2 are denoted respectively by $\mathbf{y1} = (y1_t)_{t\geq 0}$ and $\mathbf{y2} = (y2_t)_{t\geq 0}$. Let $x_{i_1}, \ldots, x_{i_{14}}$ denote the 14 initial

[1]The ECRYPT Call for stream cipher primitives states that the bit length of the IV must be 32 or 64 with a 80-bit key. Due to the recent note discussing Time Memory Trade-offs for stream ciphers `http://www.ecrypt.eu.org/stream/TMD.pdf`, the case of a 32-bit IV is not considered.

internal state bits of the LFSR that are the inputs of the functions f_1 and f_2. The sequences $\mathbf{y}_1$ and $\mathbf{y}_2$ are defined by:

$$y1_t = f(s_{i_1+t}, \ldots, s_{i_7+t}) \; ;$$

$$y2_t = f(s_{i_8+t}, \ldots, s_{i_{14}+t}) \; .$$

Each LFSR clock contributes two bits to the sequence $\mathbf{y}$ which is constructed from y_1 and y_2 as follows:

$$\mathbf{y} = y1_0, y2_0, y1_1, y2_1, y1_2, y2_2, y1_3, y2_3, \ldots \; .$$

The ABSG takes as an input the sequence $\mathbf{y} = ((y1_t, y2_t))_{t \geq 0}$. The sequence output by the ABSG is denoted by $\mathbf{z} = (z_t)_{t \geq 0}$.

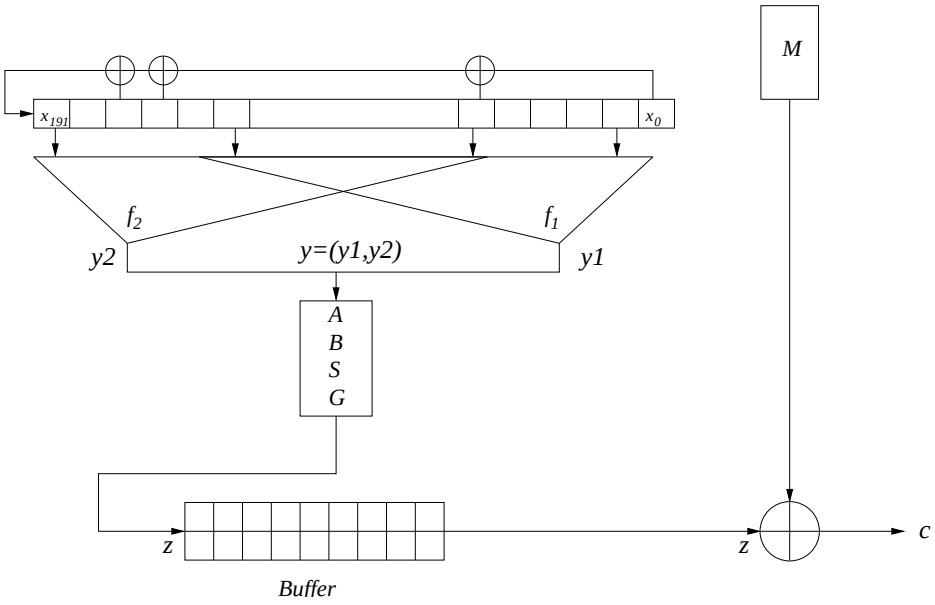

Figure 1: DECIM keystream generation

3 Specification

3.1 The filtered LFSR

This section describes the filtered LFSR that generates the sequence $\mathbf{y}$; the sequence $\mathbf{y}$ is the input of the ABSG mechanism.

The LFSR. The underlying LFSR is a maximum-length LFSR of length 192 over $\mathbf{F}_2$. It is defined by the following primitive feedback polynomial:

$$P(X) = X^{192} + X^{189} + X^{188} + X^{169} + X^{156} + X^{155} + X^{132} + X^{131} + X^{94} + X^{77} + X^{46} + X^{17} + X^{16} + X^5 + 1 \; .$$

and the recursion that corresponds to P for the LFSR is

$$\begin{aligned} s_{192+n} &= s_{187+n} \oplus s_{176+n} \oplus s_{175+n} \oplus s_{146+n} \oplus s_{115+n} \oplus s_{98+n} \oplus s_{61+n} \\ &\quad \oplus s_{60+n} \oplus s_{37+n} \oplus s_{36+n} \oplus s_{23+n} \oplus s_{4+n} \oplus s_{3+n} \oplus s_n \ . \end{aligned}$$

The filtering function f The Boolean function f which is applied for the filtering of the LFSR internal state is the symmetric function of 7 variables defined by:

$$f(x_{i_1}, \ldots, x_{i_7}) = \sum_{1 \le j < k \le 7} x_{i_j} x_{i_k}.$$

In order to increase the number of bits of the sequence $\mathbf{y}$ produced per LFSR clock, the function f is involved twice (under the name f_1 and f_2) to filter the LFSR internal state. Both filters f_1 and f_2 apply to different subsets of the LFSR state bits (see Figure 1). Namely,

- the tap positions of f_1 are 1 32 40 101 164 178 187;
- the tap positions of f_2 are 6 8 60 116 145 181 191.

In other words, the inputs of the ABSG at the stage t consist of two bits

$$\begin{aligned} y1_t &= f(s_{t+1}, s_{t+32}, s_{t+40}, s_{t+101}, s_{t+164}, s_{t+178}, s_{t+187}) \ ; \\ y2_t &= f(s_{t+6}, s_{t+8}, s_{t+60}, s_{t+116}, s_{t+145}, s_{t+181}, s_{t+191}) \ . \end{aligned}$$

3.2 Decimation

This part describes how the keystream sequence $\mathbf{z}$ is obtained from the sequence $\mathbf{y}$.

The action of the ABSG on $\mathbf{y}$ consists in splitting $\mathbf{y}$ into subsequences of the form $(\overline{b}, b^i, \overline{b})$, with $i \ge 0$ and $b \in \{0,1\}$; $\overline{b}$ denotes the complement of b in $\{0,1\}$. For every subsequence $(\overline{b}, b^i, \overline{b})$, the output bit is b for $i = 0$, and $\overline{b}$ otherwise. The ABSG algorithm is given in Figure 2.

Input: $(y_0, y_1, \ldots)$
Set: $i \leftarrow 0$; $j \leftarrow 0$;
Repeat the following steps:
1. $e \leftarrow y_i$, $z_j \leftarrow y_{i+1}$;
2. $i \leftarrow i + 1$;
3. while $(y_i = \overline{e})$ $i \leftarrow i + 1$;
4. $i \leftarrow i + 1$;
5. output z_j
6. $j \leftarrow j + 1$

Figure 2: ABSG Algorithm

3.3 Key Schedule and IV Injection

This subsection describes the computation of the initial inner state for starting the keystream generation.

3.3.1 Initial value of the LFSR state

The secret key K is a 80-bit key. The 64-bit IV is expanded to a 80-bit length vector by adding zeros from position 64 up to position 79.

The initial value[2] of the LFSR state is computed as follows.

$$x_i = \begin{cases} K_i \vee IV_i & \text{for } 0 \leq i \leq 55 \\ K_{i-56} \wedge \overline{IV_{i-56}} & \text{for } 56 \leq i \leq 111 \\ K_{i-112} \oplus IV_{i-112} & \text{for } 112 \leq i \leq 191 \end{cases}$$

The number of possible initial values of the LFSR state is $2^{2\times 56+24} = 2^{136}$. Indeed, for every $i \in [\![0; 55]\!]$, the values K_i and IV_i are uniquely determined by $K_i \vee IV_i$, $K_i \wedge \overline{IV_i}$ and $K_i \oplus IV_i$. The 24 remaining bits of K, i.e. $K_{56}, \ldots, K_{79}$, and the 8 remaining bits of IV, i.e. $IV_{56}, \ldots, IV_{63}$, are used once with an XOR operation.

3.3.2 Update of the LFSR state

The LFSR is clocked 192 times using the two following transformations.

Computation of the *non-linear feedback value* x_{191}. Let $y1_t$ (resp. $y2_t$) denote the output of f_1 (resp. f_2) at time t. Let lv_t denote the linear feedback value at time $t > 0$. Then, the value of x_{191} at time t is computed as shown in Figure 3 using the equation:

$$x_{191} = lv_t \oplus y1_t \oplus y2_t \ .$$

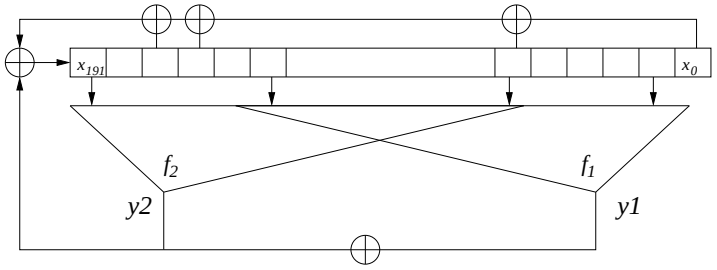

Figure 3: Computation of x_{191}

Permutations over 7 elements. After each LFSR clock, the values $y1_t$ (resp. $y2_t$) are computed in order to obtain the feedback bit. Afterwards, one of two permutations denoted π_1 and π_2 is applied on 7 elements of the current LFSR state. At each clock, two bits are input to the ABSG and if the output of the ABSG is 1, then π_1 is applied, otherwise the output of the ABSG is ε or 0 and π_2 is applied. Notice that, the probability that the ABSG outputs a bit is 2/3 and then, π_1 is applied with probability 1/3 and π_2 is applied with probability 2/3.

The permutations π_1 and π_2 are defined by:

$$\pi_1 = (1\,6\,3)(4\,5\,2\,7), \quad \pi_2 = (1\,4\,7\,3\,5\,2\,6).$$

[2]The computation of the initial value using a 64-bit IV can be adapted for 80-bit IV as explained in Section 5

The permutation is applied on the tap positions 5, 31, 59, 100, 144, 177, 186.

We sum up the keystream generation mechanism. The initial LFSR state is computed from the 80-bit key and the 64-bit IV as described in subsection 3.3.1 and the following steps are repeated 192 times:

1. computation of the non-linear feedback value;
2. application of π_1 or π_2;
3. shift of the LFSR.

3.4 Buffer mechanism

The rate of the ABSG mechanism is irregular and therefore we use a buffer in order to guarantee a constant throughput, that is, a first-in-first-out queuing technique.

For every α bits that are input into the ABSG, the buffer is supposed to output one bit exactly. If the ABSG outputs one bit when the buffer is full, then the newly computed bit is not added into the queue, i.e. it is dropped. Assuming that the initial inner state has been computed (it is denoted by $z_0, \ldots, z_{191}$), the ABSG mechanism starts at the beginning of a loop and the buffer is empty. The keystream generation process starts when the buffer is full.

We choose $\alpha = 4$ and we propose a buffer of length 32. With these parameters, the probability that the buffer is empty while it has to output one bit is less than 2^{-89}.

4 Design rationale

This section describes the design choices.

4.1 The filtered LFSR

The LFSR The length of the LFSR, which corresponds to the size of the internal state of the cipher, must be at least 160 in order to avoid time-memory-date trade-off attacks. We would like to have some security margin. Most notably, we need to deal with a reduction of the size of the potential initial state due to the initialization procedure (see Section 4.3). Therefore, a 192-bit LFSR has been chosen.

The choice of the primitive feedback polynomial P must be made in accordance with the following constraints. The differences between two consecutive positions of the inputs of the feedback polynomial are pairwise coprime. Furthermore, the weight of P must be large enough in order to prevent the existence of sparse multiples with low degree that could be exploited in fast correlation attacks or in distinguishing attacks. However, we do not want the weight of P to be too large, in order to reduce the overall computational time of the cipher.

The filtering function Since Decim is a hardware-oriented cipher, the Boolean filtering function must have a low-cost hardware implementation. Moreover, it must satisfy some well-known cryptographic properties. It has to be noticed that not all usual cryptographic criteria are necessarily relevant since the sequence $\mathbf{y}$ is the input sequence of the ABSG mechanism. First, the Boolean function f must be balanced. The Boolean function f is expected to be far from an affine function (using the Hamming distance). Furthermore, we think that the propagation criterion (PC) might be considered in order to prevent differential

attacks on the initialization vector. Then, we want to choose a Boolean function f such that most of the derivative functions $x \mapsto D_u f(x) = f(x) + f(x+u)$ are balanced. In other respects, requiring correlation-immunity seems irrelevant for filtering purposes. In order to get an efficient computation of the function, the Boolean function f has been chosen to be symmetric, i.e. the value of f only depends on the Hamming weight of the input. The symmetric Boolean functions that best fulfill the previous mentioned criteria are quadratic and have an odd number of input variables. For instance, among symmetric functions, only the quadratic ones satisfy the propagation criterion of degree greater that 1 [9, 20, 4], and quadratic symmetric functions of 7 variables possess the highest possible nonlinearity for a 7-variable function [16].

Two copies of the function are involved for filtering the LFSR internal state in order to increase the number of bits entering the ABSG at each step. The main reason is that the throughput of the cipher and the size of the buffer highly depends on this parameter.

Assuming knowledge of the keystream $\mathbf{z}$, an attacker will have to guess some bits of the sequence $\mathbf{y}$ in order to attack the function f. The knowledge of the bits of $\mathbf{y}$ directly yields to equations in the bits of the initial state of the LFSR. Thus, the number of monomials in the bits of the initial state of the LFSR that are involved in these equations has to be maximized. Moreover, this number has to grow quickly during the first clocks of the LFSR. This implies the following two conditions:

1. each difference between two positions of bits that are input to f_1 or f_2 should appear once;

2. some inputs of f_1 or f_2 should be taken at positions near the one of the feedback bit (which means that some inputs should be leftmost on Figure 1).

Finally, the tap positions of the inputs of the Boolean functions f_1 and f_2 and the inputs of the feedback relation should be independent.

4.2 Decimation

The ABSG mechanism consists of compressing the input sequence in a very simple way and it operates a highly nonlinear transformation. As an irregular decimation, it prevents algebraic attacks and some fast correlation attacks.

The best known attack on the ABSG mechanism [11] when filtering a maximum-length LFSR of length L has complexity $\mathcal{O}(2^{\frac{L}{2}})$ and requires $\mathcal{O}(L2^{\frac{L}{2}})$ keystream bits. This attack is based on the fact that, given an output bit b, the probability that b was output by the pattern bb in the input is $1/2$. A necessary condition on the sequence $\mathbf{y}$ to make this attack inefficient, that is, having a complexity greater than $\mathcal{O}(2^{80})$, is to make the linear complexity of the sequence $\mathbf{y}$ be greater than 160.

4.3 Key schedule and IV injection

The main components of the keystream generation are re-used for the key schedule and the IV injection. In addition, two small permutations (over 7 elements) are involved.

By using a 80-bit key and a 64-bit IV, the number of possible initial states is at most 2^{144}. In the present case, the number of possible LFSR initial states is 2^{136}. The key schedule includes a non-linear feedback mechanism that is repeated L times, where L is the length of

the register. Thus, in order to deal with the reduction of the potential internal state of the register during this phase, and considering that this non-linear feedback behaves randomly, we chose $L = 192$ to ensure that the final internal state was at least twice the key length, that is, 160.

4.4 The buffer mechanism

The buffer mechanism guarantees a constant throughput for the keystream. However, the buffer must have a reasonable length since the keystream generation process starts when the buffer is full.

Recall that for every α bits that are input into the ABSG, the buffer is supposed to output one bit exactly. The output rate of the ABSG is 1/3 in average. Then, the value of α is at least 3. Since each LFSR clock contributes two bits to the input sequence entering the ABSG mechanism, then we choose $\alpha = 4$. For $\alpha = 4$ and a buffer of length 32, the probability that the buffer is empty while it has to output one bit is less than 2^{-89} (the analyse of the buffer is given in Appendix A). In order to have 4 bits enter the ABSG mechanism, we suggest to use either a hardware speed-up mechanism to enter the four bits at once (see Section 6 for details) or simply to clock the LFSR twice before outputting one bit.

5 Security properties

In [13] and next in [3], Time Memory Trade-offs for stream ciphers are discussed. This kind of attacks can be applied if the state space of the cipher is too small. Then, a known necessary condition on the state size of a stream cipher is that it has to be at least twice as large as the secret key length.

We recall the results given in [3]. Let D denote the number of frames, P be the precomputation time, T be the time of the on-line attack and M be the memory which allows to recover the secret key of one frame.

Attack 1 If the IV size is smaller than half the key size, then it is possible to mount a TMD attack in which D, P, T et M are all smaller than exhaustive key search. This attack applies when using a 32-bit IV and a 80-bit key.

Attack 2 If the IV size is smaller than the key size, then it is possible to mount a TMD attack in which no restriction is imposed on the precomputation time P, and the complexities D, T and M are smaller than exhaustive key search. This attack applies when using a 64-bit IV and a 80-bit key.

The computation of the initial value of the LFSR state given in subsection 3.3.1 for 64-bit IV can be adapted for 80-bit IV. Then, using a 80-bit K and a 80-bit IV, we suggest to compute the initial value of the LFSR state as follows.

$$x_i = \begin{cases} 0 & \text{for } 0 \leq i \leq 31 \\ K_{i-32} \oplus IV_{i-32} & \text{for } 32 \leq i \leq 111 \\ K_{i-112} & \text{for } 112 \leq i \leq 191 \end{cases}$$

The number of possible initial values of the LFSR state is 2^{160}.

Since the ECRYPT call for papers states an 64-bit IV, we use the following security model.

5.1 Security model

The target security level of DECIM is 80 bits using the following security model. The attacker is a probabilistic Turing Machine with access to a black box (oracle) that accepts the following three instructions:

- RESET which generates a random key;
- INIT with a 64-bit input which initializes the internal state of the stream cipher with a new chosen IV;
- GETSTREAM with a 1-bit output which generates the next bit of the stream.

The attacker's goal is to distinguish with probability 2/3 between a black box that generates random output, and a black box that implements the stream cipher. The attacker is allowed to do 2^{80} elementary operations, an instruction to the black box being an elementary operation. This security model falls under the remarks made by Hong and Sarkar [13] on the Time/Memory trade-off attacks, because the precomputation time is not bounded by our model. We do not know of a formal security model that restricts the precomputation time, i.e. that only allows the attacker to be one of the probabilistic Turing machines that can be built in a reasonable time from the current content of today's computers. Therefore, our claim is that DECIM is secure against an attacker that is not allowed to benefit from a precomputation which is more expensive than the exhaustive search for the secret key.

5.2 Guess-and-determine attack on the ABSG

As mentioned in Section 4.2, the best known attack on the ABSG filtering a single maximum-length LFSR is based on a guess of the most favourable case. Such a guess requires ℓ output bits in order to guess 2ℓ inputs bits. The guess is correct with probability $\frac{1}{2^\ell}$. In order to check the correctness of his guess, the attacker should try to solve the equations in the bits of the initial state of the LFSR that arise from the bits of $\mathbf{y}$ he has guessed.

This attack can be used in order to reconstruct L consecutive bits of the sequence y from $\frac{L}{2}$ consecutive bits of the sequence z; it costs $\mathcal{O}(2^{\frac{L}{2}})$ and requires $\mathcal{O}(L2^{\frac{L}{2}})$ bits of z.

Let $\Lambda(y)$ denote the linear complexity of y. Then, the minimal length of a linear feedback shift register which generates the sequence y is $\Lambda(y)$. The previous attack can be used to reconstruct the initial state of the equivalent LFSR that generates the sequence y. Then, this attack costs $\mathcal{O}(2^{\frac{\Lambda(y)}{2}})$ to recover $\Lambda(y)$ consecutive bits of y.

We have checked that the linear complexity of y is the best linear complexity expected according to the choice of the Boolean function and the primitive polynomial, that is, $\Lambda(y) = 2 \times 18528 = 37056$.

5.3 Guess-and-determine attack focusing on y

The aim of this attack is to reconstruct the initial inner state by choosing, for every bit b of the sequence y, the most probable Hamming weight of the input of the Boolean function f_1 and f_2. The Hamming weight of $f^{-1}(0)$ equals 4 with probability $\frac{35}{64}$ and the Hamming weight of $f^{-1}(1)$ equals 3 the same probability, that is $\frac{35}{64}$.

At each LFSR clock, both f_1 and f_2 output one bit. Recall that we have $f = f_1 = f_2$ where f is a quadratic symmetric Boolean function and the number of monomials $X_{i,j} = x_i x_j$

of f equals 21. Then, at each LFSR clock, the number of new variables $X_{ij} = x_i x_j$ is at most 42. By considering the tap positions of f_1 and f_2, we get that one can recover 100 variables with a complexity greater that $\mathcal{O}(2^{87})$.

5.4 Distinguishing attack

The feedback polynomial has been chosen carefully, i.e. it has not low Hamming weight multiples at least for the first 2^{40} next degrees. However, we mention the possibility of a distinguishing attack similar to the distinguishing attack on the Self-Shrinking Generator given in [7].

5.5 Side channel attacks

Such attacks on keystream generators are based on exploiting for instance the time or power consumption depending on the value of the secret key and the initialization vector. Timing measurements at the output of the keystream generator is useless since a buffer is used and the throughput is constant. However, if the attacker gets timing information from the internal keystream generator, then timing attacks apply.

6 Hardware implementation

There is a trade-off between the size of the hardware implementation and the throughput of the cipher. Indeed, the 32-bit length of the buffer has been chosen to ensure that the buffer is ready with probability $(1 - 2^{-89})$ to output one bit every 4 bits entered into the ABSG.

Since each LFSR clock contributes two bits to the sequence entering the ABSG mechanism, one first solution is to clock twice the LFSR before outputting one bit. A second solution, that is studied in this section, is to increase the clock rate of the LFSR in order to produce 4 bits per clock.

The hardware implementation of the ABSG mechanism (with a single input) is low-cost (see Figure 7). However, designing an ABSG mechanism with low complexity which takes as an input a sequence of bit pairs or a sequence of bit quadruples instead of a sequence of bits is still an open problem.

Here, we describe some solutions for the hardware design (a partial description in VHDL is also supplied).

6.1 The LFSR

We propose to increase the clock rate of the LFSR. A good trade-off between the throughput and the number of gates is obtained when the LFSR is clocked twice per step. It means that our circuit computes two feedback bits, s_{t+192} and s_{t+193}, simultaneously. This can be achieved without increasing the circuit area by virtually decomposing the 192-bit LFSR F into two shift registers, F_e and F_o of length 96. The first LFSR F_e contains all even taps of F and F_o all its odd taps:

$$F_e = (s_{t+190}, \ldots, s_{t+2}, s_t) \text{ and } F_o = (s_{t+191}, \ldots, s_{t+3}, s_{t+1}) \ .$$

At each clock, both s_{t+192} and s_{t+193} are computed. Both registers F_e and F_o are shifted and s_{t+192} (resp. s_{t+193}) enters the register F_e (resp. F_o), as depicted in Figure 4 of Appendix B.

6.2 The initialization step

The initialization of the LFSR internal state with the secret key and the IV is implemented with a multiplexer and a counter (the counter aims at deciding which operation between the key and the IV must be performed).

Unfortunately, during the initialization phase, the LFSR cannot be clocked twice per step, as the internal state is permuted by π_1 and π_2 depending whether the ABSG outputs one bit or not. Therefore, the LFSR must be clocked only once per step during the initialization phase and twice per step during the keystream generation. Such a variable clock rate can be achieved at almost no cost with the previously described implementation. We only have to update each LFSR cell s_{t+i} with the output of a multiplexer, which takes the value s_{t+i+1} during the initialization and the value s_{t+i-2} during the keystream generation. A similar technique is used for updating the last 2 cells with the feedback bits. It leads to the implementation of the LFSR with variable clock rate, as described in Figure 5 of Appendix B.

Finally, both permutations π_1 and π_2 used during the initialization can be implemented easily. Actually, the LFSR cells entering the permutations do not coincide with the feedback taps and with the inputs of the filters. This design reduces the fan-in/fan-out effects.

6.3 The filter

As the clock rate of the LFSR is virtually multiplied by 2, the filter must be adapted to this acceleration. This requires to duplicate the functions in order to filter two successive virtual internal states of the LFSR simultaneously. Our hardware realization of the filtering function (see Appendix B) is based on a recent design of symmetric functions [15] which improves the generic construction [18] in the case of low-degree functions.

6.4 ABSG

Clearly, there is a trade-off between the size of the hardware implementation and the throughput of the cipher. It comes from the fact that, if we want that the buffer outputs one bit per clock, we need to implement the ABSG mechanism for a 4-bit input. But, the size of the circuit for realizing the decimation increases with the number of its inputs. We propose some partial solutions in Appendix B, but the problem of designing a 4-input ABSG with low complexity is still open.

6.5 Gate count

The number of gates involved in the proposed hardware implementation can be estimated as follows. This estimation uses the number of gates for elementary components given in [2], i.e. 12 gates for a flip-flop, 2.5 gates for an XOR, 1.5 gates for an AND and 5 gates for a MUX. Here, we have the following values for each component in the circuit:

- LFSR with variable clock rate, as described in Figure 5 of Appendix B: **3334 gates** corresponding to 192 flip-flops, 28 XORs, and 192 MUX.
- Filtering function (two copies), as described in Figure 6 of Appendix B: **74 gates** corresponding to 26 XORs and 6 ANDs.
- 1-input ABSG, as described in Figure 7 of Appendix B: **67 gates** corresponding to 2 MUX, 3 XORs, 1 AND, and 4 flip-flops.

- 4-input ABSG using a table: **6720 gates** corresponding to 70 Bytes.

7 Strengths and advantages of the primitive

The following design choices influence the miniaturization of the cipher system.

- The ABSG mechanism with 1 input bit has low-cost hardware implementation.
- The filtering function f only depends on the Hamming weight of its input and it is used twice (and not more) in order to optimize the additional cost in hardware implementation.
- The choice of the LFSR taps involved in the feedback relation, in the filtering functions and in the initialization reduces the fan-in/fan-out effects.
- The IV injection/key schedule re-uses the main components of the keystream generation mechanism.

Furthermore, each component of Decim has been chosen to get a fast stream cipher.

- The computation of the quadratic symmetric Boolean function is efficient.
- The LFSR is clocked twice per step and two copies of the filtering function are used at each clock. It allows to reduce the size of the buffer, leading to a higher throughput.

Finally, the security of Decim relies mainly on the security of the ABSG, and there is no identified attack better than exhaustive search.

Acknowledgements. The authors wish to thank Marion Videau and Matt Robshaw for helpful comments.

References

[1] F. Armknecht and M. Krause. Algebraic attacks on combiners with memory. In *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 162–176. Springer-Verlag, 2003.

[2] L. Batina, J. Lano, S.B. Örs, B. Preneel, and I. Verbauwhede. Energy, perfomance, area versus security trade-offs for stream ciphers. In *The State of the Art of Stream Ciphers: Workshop Record*, pages 302–310, Brugge, Belgium, October 2004.

[3] C. De Cannière, J. Lano, and B. Preneel. Comments on the rediscovery of Time Memory Data Tradeoffs. `http://www.ecrypt.eu.org/stream/TMD.pdf`, 2005.

[4] A. Canteaut and M. Videau. Symmetric Boolean functions. *IEEE Trans. Inform. Theory*, 2005. To appear.

[5] D. Coppersmith, H. Krawczyk, and Y. Mansour. The Shrinking Generator. In *Advances in Cryptology - CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 22–39. Springer-Verlag, 1993.

[6] N. Courtois and W. Meier. Algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer-Verlag, 2003.

[7] P. Ekdahl, T. Johansson, and W. Meier. Predicting the shrinking generator with fixed connections. In *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer-Verlag, 2003.

[8] S. Golomb. *Shift Register Sequences*. Aegean Park Press, 1982. Revised Edition.

[9] A. Gouget. On the propagation criterion of Boolean functions. In *Coding, Cryptography and Combinatorics, Progress in Computer Science and Applied Logic*, volume 23, pages 153–168. Birkhäuser Verlag, 2004.

[10] A. Gouget and H. Sibert. The Bit-Search Generator. In *The State of the Art of Stream Ciphers: Workshop Record*, pages 60–68, Brugge, Belgium, October 2004.

[11] A. Gouget, H. Sibert, C. Berbain, N. Courtois, B. Debraize, and C. Mitchell. Analysis of the Bit-Search Generator and sequence compression techniques. In *Fast Software Encryption - FSE 2005*, Lecture Notes in Computer Science. Springer-Verlag, 2005. To appear.

[12] M. Hell and T. Johansson. Some attacks on the Bit-Search Generator. In *Fast Software Encryption - FSE 2005*, Lecture Notes in Computer Science. Springer-Verlag, 2005. To appear.

[13] J. Hong and P. Sarkar. Rediscovery of Time Memory Tradeoffs. `http://eprint.iacr.org/2005/090.ps`, 2005.

[14] M. Krause. BDD-based cryptanalysis of keystream generators. In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 222–237. Springer-Verlag, 2001.

[15] C. Lauradoux. The complexity of symmetric Boolean functions. In *Ecole de Jeunes Chercheurs en Algorithmique et Calcul Formel 2005*, Montpellier, France, April 2005.

[16] S. Maitra and P. Sarkar. Maximum nonlinearity of symmetric Boolean functions on odd number of variables. *IEEE Trans. Inform. Theory*, 48(9), 2002.

[17] W. Meier and O. Staffelbach. The Self-Shrinking Generator. In *Advances in Cryptology - EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 205–214. Springer-Verlag, 1994.

[18] D.E. Muller and F.P. Preparata. Bounds to complexities of networks for sorting and switching. *Journal of the ACM*, 22:1531–1540, 1975.

[19] R.A. Rueppel. *Analysis and Design of stream ciphers*. Springer-Verlag, 1986.

[20] M. Videau. On some properties of symmetric Boolean functions. In *Proceedings 2004 IEEE International Symposium on Information Theory*, page 500. IEEE Press, 2004.

A The buffer mechanism

We analyze the buffer supposing that the inputs of the ABSG are random. Let ℓ be the length of the buffer. We assume that α bits are input to the ABSG when the buffer is supposed to output one bit exactly. What we want to do is to determine α and ℓ such that the probability that the buffer becomes empty is sufficiently low. This is similar to the method proposed in [5] in the case of the Shrinking Generator. However, the analysis is more difficult here. Indeed, for the Shrinking Generator, the probability that, at a given clock, a bit is output, does not depend on what happened beforehand, whereas in the case of the ABSG, this probability depends on the previous clock when one bit was output.

Suppose that, during the process, the buffer cannot output one bit because it is empty. This is equivalent to the existence of a pair (t, n) such that

1. at time t, the buffer contains ℓ bits, and outputs 1 bit (so that it contains $\ell - 1$ bits afterwards);
2. for every j, with $0 < j < n$, after the input of αj bits into the ABSG starting from t, the buffer never contains ℓ bits and it can always output a bit in time;
3. after the input of αn bits into the ABSG starting from time t, the buffer is empty while it has to output one bit.

We are going to compute the probability $P_{\alpha,\ell}$ that, for a given t that satisfies condition 1, there exists an n such that the pair (t, n) satisfies conditions 1, 2 and 3. Let $U_{\alpha,\ell}(n)$ be the set of all the finite sequences of length αn such that, when input to the ABSG, the output is strictly less than $n - \ell$-bit-long. Then we have:

$$P_{\alpha,\ell} = P(\exists n, 2 \text{ and } 3 \text{ are verified}).$$

Now, if for some n, conditions 2 and 3 are verified, then the finite sequence $S(n)$ consisting of the αn bits input to the ABSG from $t+1$ until $t+n$ belongs to $U_{\alpha,\ell}(n)$. Thus, we obtain:

$$P_{\alpha,\ell} \leq P(\exists n, S(n) \in U_{\alpha,\ell}(n)) \leq \sum_{n \geq 0} P(S(n) \in U_{\alpha,\ell}(n)).$$

The number of sequences of length x that produce at least y output bits when they are input to the ABSG is given by (combinatorial result):

$$N(x, y) = \sum_{p \geq y} \left[2^{p+1} \binom{x-p-1}{p} + 2^p \binom{x-p-1}{p-1} \right] .$$

Thus, we have $\#U_{\alpha,\ell}(n) = 2^{\alpha n} - N(\alpha n, n - \ell)$, and we deduce

$$P(S(n) \in U_{\alpha,\ell}(n)) \leq 1 - \frac{N(\alpha n, n-\ell)}{2^{\alpha n}}.$$

We eventually obtain

$$P_{\alpha,\ell} \leq P'_{\alpha,\ell} = \sum_{n \geq 0} (1 - \frac{N(\alpha n, n-\ell)}{2^{\alpha n}}) .$$

Suppose now that we want the buffer to output N bits. The probability $P_{\alpha,n}(N)$ that the buffer will not output one bit because it is empty before finishing is at most

$$\begin{aligned} P_{\alpha,n}(N) &= \sum_{t=0}^{N-1} P(\exists n, (t,n) \text{ satisfies 1, 2 and 3}) \\ &= \sum_{t=0}^{N-1} P(t \text{ satisfies 1}) P_{\alpha,\ell} \\ &\leq NP_{\alpha,\ell} \\ &\leq NP'_{\alpha,\ell} \ . \end{aligned}$$

These results point out that the use of a buffer with a reasonable size implies that the number of bits α entering the ABSG must not be too small.

For instance, for $k = 32$ and $\alpha = 4$, we find experimentally $2^{-90} < P'_{4,32} \sim 1.286^{-27} < 2^{-89}$. As detailed in Section 6, we suggest a hardware speed up mechanism in order to input 4 bits per real clock into the ABSG with a buffer of length 32, which consists in splitting the register and doubling the number of connections. This is enabled by the weight of the polynomial and the number of entries of the Boolean functions. Then, for N expected output bits, the probability that the buffer will get empty and not be able to output one bit before completing the task is lower than $N \cdot 2^{-89}$.

Remark 1 *In the previous computation, we did not take into account the fact that the number $N(x, y)$ is for finite sequences of length x that are input to the ABSG at the beginning of a loop. If the ABSG algorithm starts in the middle of a loop, then the first loop is shorter, and $N(x, y)$ does not apply for finite sequences of length x that are input into the ABSG in the middle of a loop. Then, in our computations, $N(x, y)$ should be replaced with a higher value. This would mean replacing $1 - \frac{N(\alpha n, n-\ell)}{2^{\alpha n}}$ with a smaller value. Therefore, all the inequalities and computations we wrote remain valid.*

B Hardware implementation

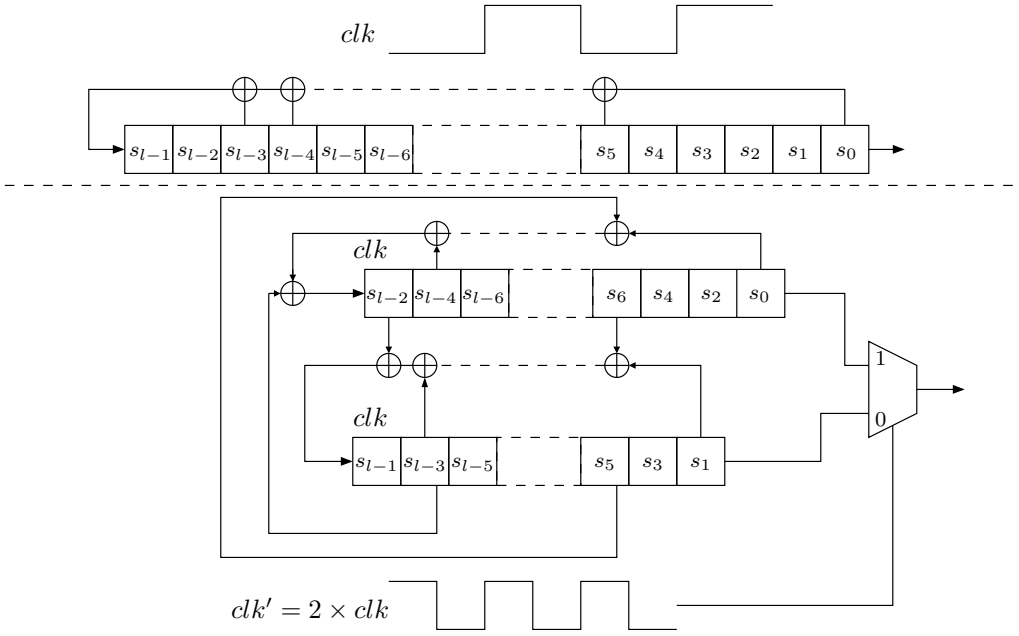

Figure 4: Equivalent representation of LFSR with clock rate virtually multiplied by 2

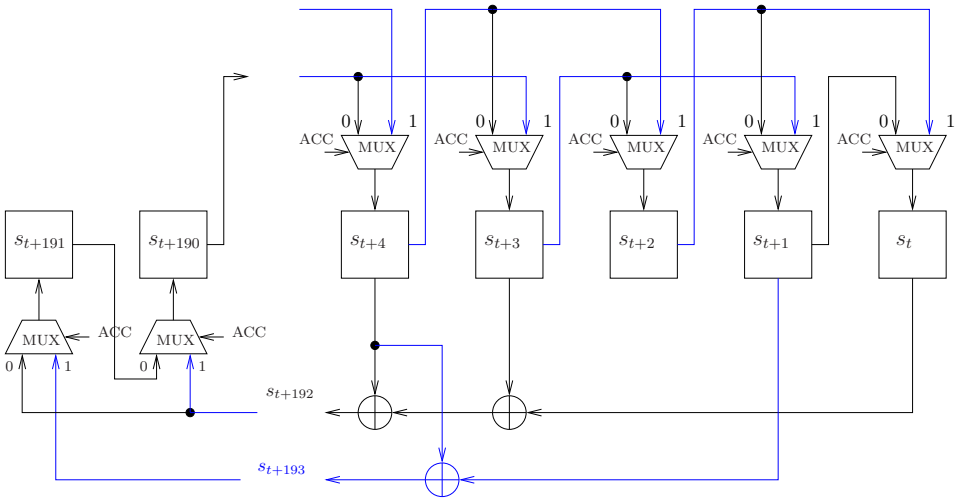

Figure 5: Implementation of the LFSR with variable clock rate: the clock rate is virtually multiplied by 2 when ACC=1.

The filter Any symmetric function of degree $d < 2^t$ for some integer t can be computed from the Hamming weight modulo 2^t of its input vector [4, 20]. For the quadratic symmetric

function of 7 variables used in DECIM,

$$f(x_1, \ldots, x_7) = \sum_{1 \leq i < j \leq 7} x_i x_j \ ,$$

we have that $f(x) = 1$ if and only if $wt(x) \bmod 4 \in \{2, 3\}$. Therefore, $f(x)$ corresponds to the 2nd bit in the binary representation of $wt(x)$.

The Hamming weight modulo 4 can be computed with 2 full-adders in a sum (s) propagate configuration (i.e. one input of a Full-adder in the stage is the s output of the previous one) and a majority function. This circuit is depicted in Figure 6.

The first full-adder computes $x_1 + x_2 + x_3 = s_1 + 2c_1$. Then, the second one provides $x_5 + x_6 + s_1 = s_2 + 2c_2$. And finally, the majority function gives

$$c_3 = MAJ(x_6, x_7, s_2) \ .$$

Obviously, c_3 corresponds to the carry bit of a full-adder. Then, it is easy to see that

$$wt(x) = x_1 + \ldots + x_7 = (x_1 \oplus \ldots \oplus x_7) + 2(c_1 + c_2 + c_3) \ .$$

Therefore, the second bit in the binary representation of $wt(x)$ corresponds to the parity of $(c_1 + c_2 + c_3)$, i.e. to the XOR between the 3 carry bits.

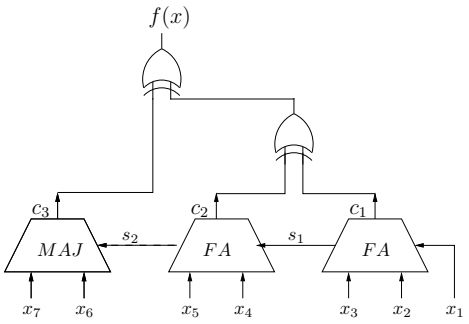

Figure 6: Boolean quadratic symmetric function

ABSG A hardware implementation of the ABSG with a single input bit is given in Figure 7. It requires 8 gates and 4 flip-flops (the complexity can even be lower if flip-flops with an enable input are available).

But, with our implementation, the ABSG mechanism receives either 2 inputs (during the initialization phase) or 4 inputs during the keystream generation. Thus, we need to have a description of ABSG which enables to perform the pattern research in an efficient way.

The ABSG with 2 input bits can be easily represented by an automaton (see the VHDL description). However, implementing the ABSG with 4 inputs is more difficult.

- The first possibility is to serialize the bits output by the filtered LFSR and to clock a 2-input ABSG at a speed rate multiplied by two compared to the speed of the filtered LFSR (or to clock a 1-input ABSG at a speed rate multiplied by four). This is easy but not always possible under important frequency limit constraints.

- A second solution consists in implementing directly the automaton which corresponds to the 4-input ABSG. This automaton is quite efficient for the 2-input ABSG, but it is quite big and complex for the 4-input version.
- Another possibility is to describing the 4-input ABSG with a table. The internal state takes 5 possible distinct values. With the previously described notation, these states correspond to:
  - `command` $= 0$;
  - `command` $= 1$, `pattern` $\in \{0, 1\}$ and `next` $\in \{\neg$`pattern`$, ?\}$, where the ? means that `pattern` corresponds to the bit that entered the ABSG at time $(t-1)$.

  Therefore, we can build a table with 4 input bits corresponding to each one of the 5 possible internal state. The output of the table consists of the new internal state of the ABSG, the number of output bits (which lies in $\{0, 1, 2\}$) and the corresponding output values. Therefore, each table outputs a 7-bit value. Thus, the 4-input ABSG can be represented by 5 tables of 14 Bytes.

The 3 memory bits respectively correspond to the first bit of the pattern, p, to the second bit in the pattern, n, and to the command, c. At time t, the bit c equals 0 if and only if the input at time t corresponds to the end of a pattern. The Boolean equations corresponding to the implementation are as follows, where $d(t)$ denotes the bit which enters the ABSG at time t.

$$\begin{aligned} p(t+1) &= [d(t) \wedge \neg c(t)] \oplus [c(t) \wedge p(t)] \\ c(t+1) &= \neg c(t) \oplus [c(t) \wedge (p(t) \oplus d(t))] \\ n(t+1) &= c(t) \wedge (p(t) \oplus d(t)) \\ out(t+1) &= c(t) \wedge (n(t) \oplus d(t)) \\ control(t+1) &= \neg\,[\neg c(t) \oplus c(t)(p(t) \oplus d(t))] \end{aligned}$$

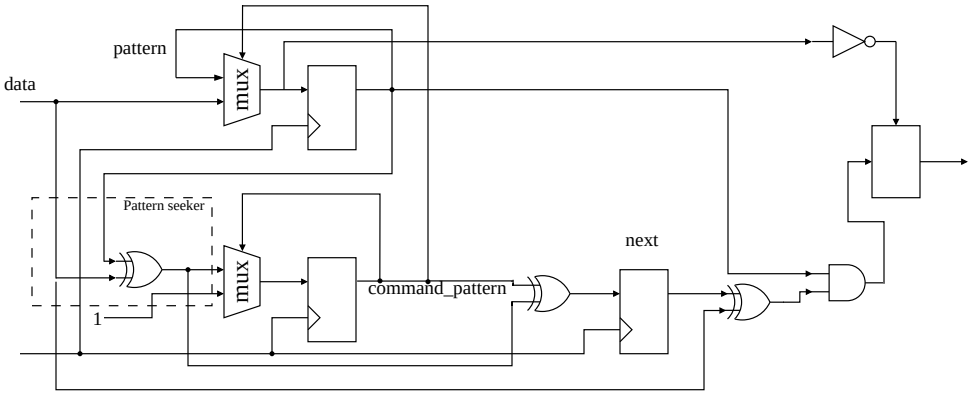

Figure 7: Hardware implementation of the ABSG