Analysis of the DVB Common Scrambling Algorithm

Ralf-Philipp Weinmann, Kai Wirt
Technical University of Darmstadt
Department of Computer Science
Darmstadt, Germany
{weinmann,kwirt}@cdc.informatik.tu-darmstadt.de

October 12, 2004

Abstract

The Common Scrambling Algorithm (CSA) is used to encrypt streams of video data in the Digital Video Broadcasting (DVB) system. The algorithm cascades a stream and a block cipher, apparently for a larger security margin. In this paper we set out to analyze the block cipher and the stream cipher separately and give an overview of how they interact with each other. We present a practical attack on the stream cipher. Research on the block cipher so far indicates it to be resistant against linear and algebraic cryptanalysis as well as simple slide attacks.

Keywords: block cipher, stream cipher, cryptanalysis, dvb, paytv

1 Introduction

The DVB Common Scrambling Algorithm is an ETSI-specified algorithm for securing MPEG-2 transport streams such as those used for digitally transmitted Pay-TV. It was adopted by the DVB consortium in May 1994, the exact origin and date of the design is unclear. Until 2002, the algorithm was only available under a Non-Disclosure Agreement from an ETSI custodian. This NDA disallowed and still disallows licensees to implement the algorithm in software for "security reasons". The little information that was then available to the public is contained in an ETSI Technical Report [Eur96] and patent applications [Bew98], [WAJ98]. This changed in the Fall of 2002, when a Windows program called FreeDec appeared which implemented the CSA in software. It was quickly reverse–engineered and details were disseminated on a web site [Pse03].

For keying the CSA, so called *control words* are used. These control words are provided by a conditional access mechanism, which generates them from encrypted control messages embedded in the transport stream. Conditional access mechanisms vary between broadcasters and can be more easily changed than the actual scrambling algorithm. Examples for commonly used conditional access mechanisms are Irdeto, Betacrypt, Nagravision, CryptoWorks etc. A new common key is usually issued every 10–120 seconds. The great relevance of CSA lies in the fact that every encrypted digital Pay-TV transmission in Europe is secured using this algorithm. A practical break of CSA would thus affect all broadcasters and could not be remedied by changing the conditional access mechanism.

The scrambling algorithm can be seen as the layering of two cryptographic primitives: a 64-bit block cipher and a stream cipher. Both ciphers employ a common key; the stream cipher uses an additional 64-bit nonce, the origin of which we will discuss later.

In this paper we investigate the two ciphers independently and show weaknesses. Although we do not present a break of the scrambling algorithm we present a known-plaintext attack on the stream cipher and show preliminary results on the block cipher.

The rest of this paper is organized as follows: Section 2 defines the notation used. In Section 3 we describe the two ciphers and how they are combined in the CSA. Our attack on the stream cipher as well as a presentation of properties of the block cipher follow in Sections 4 respectively 5. Section 6 concludes.

2 Definitions

In the rest of this paper we use the following notation:

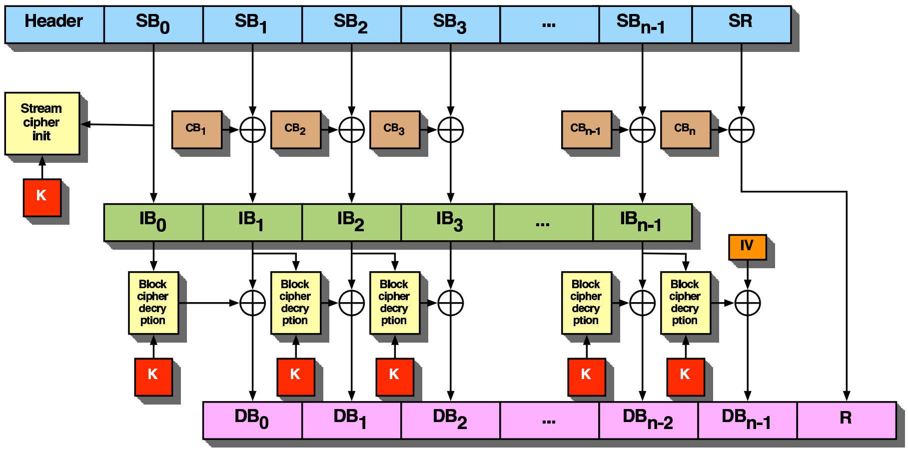
K	the common key. A 64 bit key used for both the stream and the block cipher
k_i	denotes the i-th bit of K
K^E	denotes the expanded key which is derived through the key schedule of the block cipher
SB_i	is the i-th 8-byte block of the scrambled data stream SB_0 is used as nonce in the stream cipher
CB_i	is the i-th 8-byte block of stream cipher output
IB_i	intermediate blocks. See Figure 1 for details.
DB_i	is the i-th 8-byte block of descrambled data
R	denotes the residue and
SR	is used for the scrambled residue
IV	an initialization vector. Always equals the zero block.
rol	bitwise rotation to the left by one bit
$\|\|$	denotes concatenation
t_i	state of the stream cipher after i clocks t_{-31} is the starting state, t_0 the state after the initialization
I^A	is an additional input for the stream cipher generated from the nonce
l_w	denotes the cycle length, i.e. the smallest number $j - i$ for which $t_j = t_i$
l_s	is the length of a small cycle, i.e. the smallest number $j - i$ for which the feedback shift register 1 has the same value in t_j and t_i

3 Description

3.1 Cascading the block and the stream cipher

The scrambling algorithm can be seen as a cascade of a block cipher and a stream cipher. Both ciphers use the same 64-bit key K, which is called the

Figure 1: Combination of block- and stream cipher.



common key. We will now describe how the block and the stream cipher are combined. Figure 1 depicts the descrambling process.

For scrambling the payload of a m-byte packet, it is divided into blocks (DB_i) of 8 bytes each. If an adaption field was used, it is possible that the length of the packet is not a multiple of 8 bytes. Thus the last block is $n < 8$ bytes long and shall be called *residue*.

The sequence of 8-byte blocks is encrypted in reverse order with the block cipher in CBC mode, whereas the residue is left untouched. The last output of the chain IB_0 is then used as a nonce for the stream cipher. The first $m - 8$ bytes of keystream generated by the stream cipher are XORed to the encrypted blocks $(IB_i)_{i \geq 1}$ followed by the residue.

3.2 The stream cipher

3.2.1 Overview

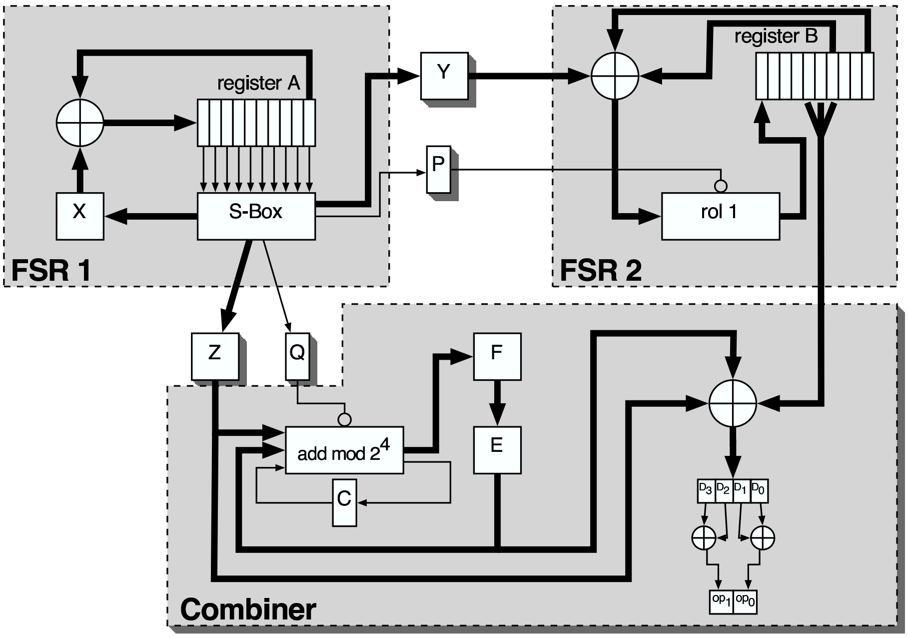
The stream cipher is built of two feedback-shift-registers and a combiner with memory. The overall layout is shown in Figure 2. The registers p, q and c are bit registers. All other registers are 4 bit wide.

The stream cipher operates in one of two modes. The first one is the initialization mode in which the starting state of the cipher is set up. The second one is the generating mode in which the cipher produces two pseudo-random bits per clock cycle.

3.2.2 Key Schedule

The cipher uses the common key K and the first scrambled block of the transport stream SB_0 as a nonce to set up the initial state. At first all registers of the cipher are set to 0. Then the common key $K = k_0, \dots, k_{63}$ is loaded into the shift registers $A := a_{0,j}, \dots, a_{9,j}$ and $B := b_{0,j}, \dots, b_{9,j}$ with $0 \leq j \leq 3$ according to the following rule:

Figure 2: The stream cipher.



$$a_{i,j} = \begin{cases} k_{4\cdot i+j} & i \leq 7 \\ 0 & \text{else} \end{cases}$$

$$b_{i,j} = \begin{cases} k_{32+4\cdot i+j} & i \leq 7 \\ 0 & \text{else} \end{cases}$$

In the following a_i and b_i denote the 20 4-bit registers $a_{i,0}, \ldots, a_{i,3}$ and $b_{i,0}, \ldots, b_{i,3}$ respectively.

Hereafter the cipher is in initialization mode. It uses SB_0 and the feedback register D as input and performs 32 clock cycles to calculate the starting state. The inputs for feedback shift registers 1 and 2 are derived from SB_0:

$$(I^A, I^B) := \begin{cases} (SB_0 \text{ div } 2^4, SB_0 \text{ mod } 2^4) & \text{in state } t_i, i \in \{-31, -29, \ldots\} \\ (SB_0 \text{ mod } 2^4, SB_0 \text{ div } 2^4) & \text{else} \end{cases}$$

Thus in every odd cycle number I^A is the high nibble of SB_0 whereas I^B is the low nibble. In even cycles the nibbles are used the other way round. See below for the equations which update the internal cipher state.

3.2.3 Generation mode

Feedback shift register 1 The feedback a_0' of shift register A is calculated as

$$a_0' := \begin{cases} a_9 \oplus X & \text{if not in init mode} \\ a_9 \oplus X \oplus D \oplus I^A & \text{else} \end{cases}$$

The next value A' for register A is then given by

$$A' \quad := \quad (a'_0, a_0, \ldots, a_8)$$

Feedback shift register 2 The feedback b'_0 of shift register B is given by

$$b'_0 \quad := \quad \begin{cases} b_6 \oplus b_9 \oplus Y & \text{if not in init mode} \\ b_6 \oplus b_9 \oplus Y \oplus I^B & \text{else} \end{cases}$$

and the new value B' for B is

$$B' \quad := \quad \begin{cases} (b'_0, b_0, \ldots, b_8) & p = 0 \\ (rol(b'_0), b_0, \ldots, b_8) & \text{else} \end{cases}$$

Other registers New values for the other registers, namely X, Y, Z, p and q are derived from seven 5×2 S-Boxes. Table 1 shows which bits from shift-register A are used as input for the S-Boxes and how the new register values are constructed. The S-Boxes itself are shown in Table 4. Table 6 gives an algebraic description of the S-Boxes, with a being the most significant input bit and e the least significant.

Combiner The stream cipher uses a combiner with memory to calculate two bits of output per clock. The memory of the combiner consists of registers E, F and c. In each cycle a new state for these registers is determined according to

$$(E, F)' \quad := \quad \begin{cases} (F, E) & q = 0 \\ (F, E + Z + c \bmod 2^4) & \text{else} \end{cases}$$

c is unchanged if $q = 0$. Otherwise it is 1 if $E + Z + c \geq 2^4$ and 0 else.

The output of the generator is calculated by $D_2 \oplus D_3 || D_0 \oplus D_1$ where $D := E \oplus Z \oplus B^{out}$ with B^{out} given by

$$\begin{aligned} B_3^{out} &:= b_{2,0} \oplus b_{5,1} \oplus b_{6,2} \oplus b_{8,3} \\ B_2^{out} &:= b_{5,0} \oplus b_{7,1} \oplus b_{2,3} \oplus b_{3,2} \\ B_1^{out} &:= b_{4,3} \oplus b_{7,2} \oplus b_{3,0} \oplus b_{4,1} \\ B_0^{out} &:= b_{8,2} \oplus b_{5,3} \oplus b_{2,1} \oplus b_{7,0} \end{aligned}$$

Table 1: S-Box input and generation of new register values.

S_1	$a_{3,0}$	$a_{0,2}$	$a_{5,1}$	$a_{6,3}$	$a_{8,0}$
S_2	$a_{1,1}$	$a_{2,2}$	$a_{5,3}$	$a_{6,0}$	$a_{8,1}$
S_3	$a_{0,3}$	$a_{1,0}$	$a_{4,1}$	$a_{4,3}$	$a_{5,2}$
S_4	$a_{2,3}$	$a_{0,1}$	$a_{1,3}$	$a_{3,2}$	$a_{7,0}$
S_5	$a_{4,2}$	$a_{3,2}$	$a_{5,0}$	$a_{7,1}$	$a_{8,2}$
S_6	$a_{2,1}$	$a_{3,1}$	$a_{4,0}$	$a_{6,2}$	$a_{8,3}$
S_7	$a_{1,2}$	$a_{2,0}$	$a_{6,1}$	$a_{7,2}$	$a_{7,3}$

X	$S_{4,0}$	$S_{3,0}$	$S_{2,1}$	$S_{1,1}$
Y	$S_{6,0}$	$S_{5,0}$	$S_{4,1}$	$S_{3,1}$
Z	$S_{2,0}$	$S_{1,0}$	$S_{6,1}$	$S_{5,1}$
p	$S_{7,1}$			
q	$S_{7,0}$			

3.3 The block cipher

CSA employs an iterated block cipher that operates bytewise on 64-bit blocks of data and uses a 64-bit key, the common key K. Each round of the cipher employs the same round transformation ϕ, which takes an 8-byte vector along with a single byte of the expanded key as input and outputs an 8-byte vector. This round transformation is applied 56 times. One could also lump together 8 successive rounds of the cipher into a round function ϕ' and describe a 7-round cipher which uses 64-bit subkeys; however we feel that the description we give below is more natural and easier to comprehend.

3.3.1 The key schedule

Let ρ be the bit permutation on 64-bit strings which is defined in Table 2. The expanded key $K^E = (k_0^E, \ldots, k_{447}^E)$ consists of a total of 448 bits which are recursively computed as follows:

$$\begin{aligned} k^E_{0,\ldots,63} &= k_{0,\ldots,63} \\ k^E_{64i,\ldots,64i+63} &= \rho(k^E_{64(i-1),\ldots,64i-1}) \oplus \mathtt{0x0}i\mathtt{0}i\mathtt{0}i\mathtt{0}i\mathtt{0}i\mathtt{0}i\mathtt{0}i\mathtt{0}i \quad 1 \leq i \leq 6 \end{aligned}$$

where the expression $\mathtt{0x0}i\mathtt{0}i\mathtt{0}i\mathtt{0}i\mathtt{0}i\mathtt{0}i\mathtt{0}i\mathtt{0}i$ is to be interpreted as a hexadecimal constant. We note that the key schedule is entirely $GF(2)$-linear.

3.3.2 The round function

At the core of the round transformation ϕ are the nonlinear functions f and f'. These are distinct permutations on the set of all byte values and can be seen as the S-Boxes of the cipher. Both permutations have maximum cycle length and are related to each other by a bit permutation σ, i.e. $f' = \sigma \circ f$. This bit permutation maps bit 0 to 1, bit 1 to 7, bit 2 to 5, bit 3 to 4, bit 4 to 2, bit 5 to 6, bit 6 to 0 and bit 7 to 3. See Table 5 for the actual values described by f.

Let $S = (s_0, \ldots, s_7)$ be the vector of bytes representing the internal state of the block cipher in an arbitrary round. The function ϕ taking the internal state S from round i to round $i+1$ can then be defined as

$$\begin{aligned} \phi(s_0, \ldots, s_7, k) = \; & (s_1, s_2 \oplus s_0, s_3 \oplus s_0, s_4 \oplus s_0, \\ & s_5, s_6 \oplus f'(k \oplus s_7), s_7, s_0 \oplus f(k \oplus s_7)) \end{aligned}$$

Table 2: Key bit permutation.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho(i)$	17	35	8	6	41	48	28	20	27	53	61	49	18	32	58	63
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\rho(i)$	23	19	36	38	1	52	26	0	33	3	12	13	56	39	25	40
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$\rho(i)$	50	34	51	11	21	47	29	57	44	30	7	24	22	46	60	16
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$\rho(i)$	59	4	55	42	10	5	9	43	31	62	45	14	2	37	15	54

whereas for decrypting a block of ciphertext we need the inverse function:

$$\begin{aligned}\phi^{-1}(s_0,\ldots,s_7,k) \quad = \quad & (s_7 \oplus f(s_6 \oplus k), s_0, \\ & s_7 \oplus s_1 \oplus f(s_6 \oplus k), s_7 \oplus s_2 \oplus f(s_6 \oplus k), \\ & s_7 \oplus s_3 \oplus f(s_6 \oplus k), s_4, s_5 \oplus f'(s_6 \oplus k), s_6)\end{aligned}$$

3.3.3 Encryption/Decryption

Encrypting a plaintext $P = (p_0, \ldots, p_7)$ is accomplished by

$$\begin{aligned} S^0 &= P \\ S^r &= \phi(S^{r-1}, (k^E_{8r}, \ldots, k^E_{8r+7})) \qquad 1 \leq r \leq 56 \\ C &= S^{56} \end{aligned}$$

which yields the ciphertext $C = (c_0, \ldots, c_7)$. For decrypting this ciphertext the following sequence of operations needs to be carried out:

$$\begin{aligned} S^0 &= C \\ S^r &= \phi^{-1}(S^{r-1}, (k^E_{448-8r}, \ldots, k^E_{455-8r})) \qquad 1 \leq r \leq 56 \\ P &= S^{56} \end{aligned}$$

4 Analysis of the stream cipher

In the following we denote with t_0 the stream cipher's state after the initialization. That means t_{-31} is the initial state, in which the common key is loaded in the registers A and B respectively. Given this notation we define a full cycle to be the smallest number $l_w := j - i$ for which the values of all registers in state t_i are equal to the values in t_j. Also we define a small cycle to be the smallest number $l_s := j - i$ when the values of X and A in state t_i are equal to the values in t_j.

4.1 Observation

The CSA stream cipher's state consists of 103 bits. This means that the maximum period length is 2^{103}. For cryptographic purposes, one would expect the cycle to go through a minimum of 2^{80} states. Using Floyd's cycle-finding algorithm however, we observed that after a relatively short preperiod there exist only a few different cycle lengths for different key/nonce combinations; all of these have a length of $l_w < 10^9$, which of course is much smaller than 2^{80}. When comparing the set of states in several cycles with the same length which where generated by different key/nonce pairs, one notices that these are disjunct; many different cycles with length l_w exist.

On the other hand, taking only A and X in account shows that if two cycles have the same length l_w then l_s is equal too. Moreover the sequence of states in feedback-shift-register 1 is equal. This means that if l_w is equal for two cycles then the registers A and X for these cycles are going through the same values.

We conducted a total of 10^5 experiments with random key/nonce pairs to determine the most probable period lengths for the state transition function operating on register A. Table 3 shows some small cycle lengths l_s together with the number of times $n(l_s)$ we observed a cycle of this length in our test and $a(l_s)$ the average length of the pre-period for a given cycle length.

Table 3: Probability distribution for small cycles.

$n(l_s)$	l_s	$a(l_s)$
36106	22778	152854.6
24196	97494	83098.3
18054	121992	27726.2
15171	42604	65556.8
3244	25802	17643.8
1495	108	21051.6
131	2391	3138.5

In 1.6% of all cases we observed cycle lengths not listed in the above table. For each of these the probability of occurrence must be lower than 0.2%. This observation leads to the following attack:

Calculate a table T with the states of the small cycles
for every state in T **do**
    Test if the state is correct
    Reconstruct the remaining registers
end for

It remains to show how one can determine if the state is correct and how the remaining registers can be reconstructed.

4.2 Finding the correct value for FSR1

The trivial method of finding the correct value for FSR1 is to simply try all possible values. That means that one searches through all states which belong to one of the small cycles. Summing up the number of states in Table 3 shows that in 98.4% of all cases testing $313,169$ possibilities is sufficient; this is far less than the 2^{44} possible values for A and X.

4.3 Reconstructing the remaining registers

The stream cipher's output is calculated by XORing Z, E and B^{out}. Since we can now consider A to be known, Z is fully determined. For all possible 2^9 values of E, F and c do the following:
Consider all bits of B at clock cycle t as variables with values in $GF(2)$. Generate a system of equations describing the two output bits at clock cycle $t+k$ as linear equations of bits of these variables. This system is linear since the additional inputs for the feedback shift register are fully determined by A and hence are known. In other words: for every state of A a system of linear equations that fully describes B with respect to B^{out} exists. Therefore this system can be efficiently solved using Gaussian elimination. If the system is inconsistent then the guess for E, F and c was wrong and has to be altered.

The last step of the attack is to determine which of the possible solutions for the linear equations system is the correct one. This has to be done because different values for E, F and c may lead to a solution of the system. The correct value can be determined simply by running the keystream generator with the calculated state and checking if the output corresponds to the actual output of the generator.

4.4 Results

Some of the generated equations are linearly dependent. Experimentally we derived that for finding a unique solution to the system described above, 60 equations are sufficient.

For carrying out the attack one thus needs to solve approximately $2^{19} \cdot 2^9 = 2^{28}$ systems of linear equations, each of which contains 60 equations in 40 unknowns. Experiments showed that this can be done in less than an hour on a 1.25 GHz PowerPC G4. We stress that our attack leaves much room for improvement. It might be possible to increase our chances at guessing the correct value for A from statistical deviation in the output of the stream cipher. But already our unoptimized version shows that the stream cipher can be broken in a very short time. Also, this attack is well suited to parallelization.

5 Analysis of the block cipher

We note that the round function ϕ is a weak permutation. Given the inputs x_1, x_2 and outputs $y_1 = \phi(x_1, k)$ and $y_2 = \phi(x_2, k)$ of a single round it is trivial to determine the round subkey k. The key schedule however seems to make the cipher resistant against slide attacks [BW99].

5.1 Linear approximation of the S-Boxes

The maximum bias of both S-Boxes is $\frac{17}{128}$. Trying to find a linear path through several rounds of the cipher we see that the number of active S-Boxes in the path increases exponentially in the number of rounds. Because of this fact and the high number of rounds, the authors believe that classical linear cryptanalysis poses not threat to the cipher.

5.2 Polynomial interpolation of the S-Boxes

We have interpolated the S-Boxes as polynomials over fields $GF(2^8) = \frac{GF(2)[X]}{m(X)}$ for all $m \in GF(2)[X]$ with $\deg(m) = 8$ and m irreducible. The resulting polynomials are all dense and of maximum degree. Interpolating bit traces of the S-Boxes results in polynomials consisting of 117–137 terms. Two of them are of degree 8, the other 6 of degree 7.

Thus we conclude that both representations are not useful for algebraic cryptanalysis of the cipher.

6 Conclusion

In this paper we described the Common Scrambling Algorithm and presented an analysis of the underlying stream and block cipher parts. We showed that the stream cipher is weak and can be efficiently broken. We also pointed out some properties of the block cipher which eventually could be used in an successful attack. However, since the block cipher uses 56 rounds we believe that such an attack would have to use sophisticated techniques.

Cryptanalyzing both stream and block cipher at the same time seems to be a task too daunting to attempt. Finding special cases where plaintext and corresponding ciphertext can be obtained that is encrypted with only one of the ciphers facilitates easier cryptanalysis. For the stream cipher these are packets with a residue. A sufficiently long adaption field on the other hand can lead to packets which are only protected by the block cipher.

We believe that extending the attack on the stream cipher to a key recovery is not a trivial task. Since the state update function of the stream cipher is irreversible and nonlinear, the only option we see at this point for recovering the key is to solve a large system of polynomial equations for different nonces and key streams. The nonlinear equations in this system are of the form seen in Table 6.

There are various directions for future research on these topics. First of all the attack presented offers room for further improvements like the reduction of the necessary register guesses. Investigating how to recover the Common Key K from a known state of the stream cipher is another logical step. Finally, the block cipher needs more scrutiny.

7 Appendix

References

[Bew98] Simon Bewick. Descrambling DVB data according to ETSI common scrambling specification. UK Patent Applications GB2322994A / GB2322995A , 1998.

[BW99] Alex Biryukov and David Wagner. Slide attacks. In Lars Knudsen, editor, *Fast Software Encryption: 6th International Workshop, FSE'99, Rome, Italy, March 1999. Proceedings*, volume 1663 of *Lecture Notes in Computer Science*, pages 245–. Springer-Verlag Heidelberg, 1999.

[Eur96] European Telecommunications Standards Institute. ETSI Technical Report 289: Support for use of scrambling and Conditional Access (CA) within digital broadcasting systems, 1996.

[Gol67] Solomon W. Golomb. *Shift Register Sequences*. Holden-Day San Francisco, 1967.

[Pse03] Pseudonymous authors. CSA – known facts and speculations, 2003. `http://csa.irde.to`.

[Rue86] Rainer A. Rueppel. *Analysis and design of stream ciphers*. Springer-Verlag New York, Inc., 1986.

[WAJ98] Davies Donald Watts, Rix Simon Paul Ashley, and Kuehn Gideon Jacobus. System and apparatus for blockwise encryption and decryption of data. US Patent Application US5799089 , 1998.

Table 4: S-Boxes of the stream cipher.

Input	S_1	S_2	S_3	S_4	S_5	S_6	S_7
00000	10	11	10	11	10	00	00
00001	00	01	00	01	00	01	11
00010	01	00	01	10	00	10	10
00011	01	10	10	11	01	11	10
00100	10	10	10	00	11	01	11
00101	11	11	11	10	10	10	00
00110	11	11	11	01	11	10	00
00111	00	00	01	10	10	00	01
01000	11	01	01	01	00	00	11
01001	10	11	01	10	01	01	00
01010	10	10	00	00	11	11	01
01011	00	01	11	01	11	00	11
01100	01	00	11	11	01	10	01
01101	01	00	00	00	00	11	10
01110	00	01	10	00	10	01	10
01111	11	10	00	11	01	11	01

Input	S_1	S_2	S_3	S_4	S_5	S_6	S_7
10000	00	11	01	01	10	10	01
10001	11	01	11	00	11	11	00
10010	11	00	00	11	10	00	11
10011	00	11	01	01	00	10	11
10100	10	11	11	10	00	11	00
10101	10	10	00	11	11	00	01
10110	01	00	10	00	01	01	01
10111	01	10	10	11	01	01	10
11000	10	00	10	00	01	10	10
11001	10	00	00	11	00	01	11
11010	00	01	01	10	11	01	01
11011	11	10	10	00	10	10	00
11100	01	10	00	01	11	00	10
11101	01	01	11	10	01	11	11
11110	11	11	11	10	00	11	00
11111	00	01	01	01	10	00	10

Table 5: S-Box of the block cipher. Output arranged row-wise; lower nibble on horizonal, upper on vertical.

	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0x00	0x3A	0xEA	0x68	0xFE	0x33	0xE9	0x88	0x1A	0x83	0xCF	0xE1	0x7F	0xBA	0xE2	0x38	0x12
0x01	0xE8	0x27	0x61	0x95	0x0C	0x36	0xE5	0x70	0xA2	0x06	0x82	0x7C	0x17	0xA3	0x26	0x49
0x02	0xBE	0x7A	0x6D	0x47	0xC1	0x51	0x8F	0xF3	0xCC	0x5B	0x67	0xBD	0xCD	0x18	0x08	0xC9
0x03	0xFF	0x69	0xEF	0x03	0x4E	0x48	0x4A	0x84	0x3F	0xB4	0x10	0x04	0xDC	0xF5	0x5C	0xC6
0x04	0x16	0xAB	0xAC	0x4C	0xF1	0x6A	0x2F	0x3C	0x3B	0xD4	0xD5	0x94	0xD0	0xC4	0x63	0x62
0x05	0x71	0xA1	0xF9	0x4F	0x2E	0xAA	0xC5	0x56	0xE3	0x39	0x93	0xCE	0x65	0x64	0xE4	0x58
0x06	0x6C	0x19	0x42	0x79	0xDD	0xEE	0x96	0xF6	0x8A	0xEC	0x1E	0x85	0x53	0x45	0xDE	0xBB
0x07	0x7E	0x0A	0x9A	0x13	0x2A	0x9D	0xC2	0x5E	0x5A	0x1F	0x32	0x35	0x9C	0xA8	0x73	0x30
0x08	0x29	0x3D	0xE7	0x92	0x87	0x1B	0x2B	0x4B	0xA5	0x57	0x97	0x40	0x15	0xE6	0xBC	0x0E
0x09	0xEB	0xC3	0x34	0x2D	0xB8	0x44	0x25	0xA4	0x1C	0xC7	0x23	0xED	0x90	0x6E	0x50	0x00
0x0A	0x99	0x9E	0x4D	0xD9	0xDA	0x8D	0x6F	0x5F	0x3E	0xD7	0x21	0x74	0x86	0xDF	0x6B	0x05
0x0B	0x8E	0x5D	0x37	0x11	0xD2	0x28	0x75	0xD6	0xA7	0x77	0x24	0xBF	0xF0	0xB0	0x02	0xB7
0x0C	0xF8	0xFC	0x81	0x09	0xB1	0x01	0x76	0x91	0x7D	0x0F	0xC8	0xA0	0xF2	0xCB	0x78	0x60
0x0D	0xD1	0xF7	0xE0	0xB5	0x98	0x22	0xB3	0x20	0x1D	0xA6	0xDB	0x7B	0x59	0x9F	0xAE	0x31
0x0E	0xFB	0xD3	0xB6	0xCA	0x43	0x72	0x07	0xF4	0xD8	0x41	0x14	0x55	0x0D	0x54	0x8B	0xB9
0x0F	0xAD	0x46	0x0B	0xAF	0x80	0x52	0x2C	0xFA	0x8C	0x89	0x66	0xFD	0xB2	0xA9	0x9B	0xC0

Table 6: Algebraic description of the S-Boxes used in the stream cipher.

$$
\begin{aligned}
S_{1,0} &= abce + abc + abd + bde + ab + ae + be + ce + b + d \\
S_{1,1} &= abcd + abde + abc + abd + acd + ade + bcd + bce + \\
&\quad ab + ac + bc + bd + be + cd + ce + de + a + d + e + 1 \\
S_{2,0} &= abce + abde + ade + bce + bde + ab + ac + ce + c + d + 1 \\
S_{2,1} &= abde + abc + abd + abe + acd + cde + cd + ce + b + d + e + 1 \\
S_{3,0} &= ce + de + a + b + d \\
S_{3,1} &= abcd + acde + abe + ac + abc + acd + ace + ade + bcd + bde + \\
&\quad cde + ad + bc + bd + be + cd + ce + a + b + d + e + 1 \\
S_{4,0} &= abcd + abde + acde + abc + abe + bde + ab + ad + ae + bc + \\
&\quad be + de + c + d + 1 \\
S_{4,1} &= abcd + abde + acde + abc + abe + bcd + cde + ad + ab + ae + \\
&\quad de + a + b + c + e + 1 \\
S_{5,0} &= abde + acde + acd + abe + abd + ace + bce + cde + ab + ac + \\
&\quad ae + bd + be + ce + de + c \\
S_{5,1} &= abcd + abce + acde + abd + abe + acd + bcd + bce + \\
&\quad bde + cde + ac + ad + ae + be + cd + ce + de + b + d + e + 1 \\
S_{6,0} &= abcd + abde + acde + acd + ade + bcd + cde + bc + bd + cd + \\
&\quad c + e \\
S_{6,1} &= abe + ade + bce + bde + bc + ce + a + d \\
S_{7,0} &= abde + abd + cde + bc + cd + de + a + b + c + e \\
S_{7,1} &= abcd + abdebc + acde + acd + ade + bde + ac + ae + de + \\
&\quad b + c + d + e
\end{aligned}
$$