Cascade Jump Controlled Sequence Generator (CJCSG)

Tor Helleseth[1], Cees J.A. Jansen[2] and Alexander Kholosha[1]

[1] The Selmer Center
University of Bergen, P.O. Box 7800,
5020 Bergen, Norway
[2] Banksys NV
Haachtsesteenweg 1442
1130 Brussels, Belgium
cja@iae.nl; {Tor.Helleseth,Alexander.Kholosha}@ii.uib.no

1 Introduction

Linear feedback shift registers (LFSR) are known to allow fast implementation and produce sequences with large period and good statistical properties (if the feedback polynomial is chosen appropriately). But inherent linearity of these sequences results in susceptibility to algebraic attacks. That is the prime reason why LFSR's are not used directly for key-stream generation. A well-known method for increasing the linear complexity preserving at the same time a large period and good statistical properties, is to apply clock control, i.e. to irregularly step the LFSR through successive states. Key-stream generators based on regularly clocked LFSR's are susceptible to basic and fast correlation attacks. Using irregular clocking reduces the danger from correlation attacks and provides practical immunity against fast correlation attacks.

Due to the multiple clocking, key stream generators that use clock-controlled LFSRs have decreased rate of sequence generation since such generators are usually stepped a few times to produce just one bit of the key stream. The efficient way to let an LFSR move to a state that is more than one step further but without having to step though all the intermediate states (so called, jumping) was suggested in [1]. Further in Section 2 we give a brief description of the this technique.

The extremely serious weakness found in key stream generators that use irregular clocking is their vulnerability to timing and power attacks. This was one of the reasons why the stream ciphers such as SOBER-t16 and SOBER-t32 did not pass the security evaluation and were not included into the NESSIE portfolio of strong cryptographic primitives. Using jump controlled LFSRs instead of the traditional clock-controlled ones allows to build efficient countermeasures against the side-channel attacks while preserving all the advantages of irregular clocking.

We are planning to make a word-oriented stream cipher based on the ideas of Jump Control. Theoretical basis for such an arrangement is partly developed now (see [2]). This will be implemented in a second version of the CJCSG.

We state that there are no hidden weaknesses in the key stream generator that are inserted by the designers. Security of the CJCSG is not less than the complexity of the exhaustive key search.

People from the Selmer Center who also contributed to this project are Igor Semaev, Matthew G. Parker and Håvard Raddum. We want to thank Sondre Ronjom from the Department of Informatics at the University of Bergen for making the alternative implementation of the algorithm.

2 Jump Controlled LFSR

The ideas presented in this section are well described in [1, 3, 4, 2] and were presented at SASC 2004, the Benelux Information Theory Symposium 2005 and earlier at RECSI 2002 and EIDMA Cryptography Working Group meeting in February 2003.

Consider an autonomous Linear Finite State Machine (LFSM), not necessarily an LFSR, defined by the transition matrix A of size L over GF(2) with primitive characteristic polynomial $f(x) = \det(xI+A)$, where I is the identity matrix. It is well known that A is similar to the companion matrix of $f(x)$, i.e., there exists a nonsingular matrix M such that $M^{-1}AM = S(f)$. Let z_t $(t = 0, 1, 2, \ldots)$ denote the inner state of the LFSM at stage t. Then $z_t = z_0A^t = z_0MS(f)^tM^{-1}$ and $z_tM = (z_0M)S(f)^t$. Thus, LFSMs defined by A and $S(f)$ are equivalent.

Take a matrix representation of the elements of the finite field GF(2^L). Since $f(S(f)) = 0$ and $f(x)$ is primitive, $S(f)$ can play the role of a root of f that is a primitive element in GF(2^L). Then $S(f)+I$ being an element of GF(2^L) is equal to $S(f)^J$ for some power J and, thus, $A^J = MS(f)^JM^{-1} = MS(f)M^{-1} + I = A+I$. Note that identity $S(f)^J = S(f)+I$ is equivalent to $x^J \equiv x+1 \pmod{f(x)}$ and, therefore, such a value of J is called the *Jump Index* of f. It is important to observe here that changing the transition matrix of the LFSM from A to $A + I$ results in making J steps through the state space of the original LFSM.

Let $f^\perp(x)$ denote the characteristic polynomial of the modified transition matrix $A+I$ that is equal to $f^\perp(x) = \det(xI+A+I) = f(x+1)$. The polynomial $f^\perp(x)$ is called the *dual* of $f(x)$. It is easy to see that $f(x)$ is irreducible if and only if $f^\perp(x)$ is irreducible (however, this equivalence does not hold for being primitive). It can also be shown (see [3, Theorem 2]) that if the dual polynomial $f^\perp$ is primitive (the jump index of $f^\perp$, naturally, exists) then the jump index of f is coprime with $\lambda = 2^L - 1$ and $J^\perp \equiv J^{-1} \pmod{\lambda}$.

The transition matrix A that defines the LFSM used in the CJCSG has a very special form, i.e.:

$$\begin{pmatrix} d_L & 0 & 0 & \cdots & 0 & 1 \\ 1 & d_{L-1} & 0 & \cdots & 0 & t_{L-1} \\ 0 & 1 & d_{L-2} & \ddots & \vdots & \vdots \\ 0 & 0 & \ddots & \ddots & 0 & \vdots \\ \vdots & \vdots & \ddots & 1 & d_2 & t_2 \\ 0 & 0 & \cdots & 0 & 1 & d_1 + t_1 \end{pmatrix} \tag{1}$$

It is the companion matrix of a polynomial of degree L (L is even) with additional $L/2$ ones on the main diagonal. The right-hand column has constants t_i, $1 \leq i \leq L-1$, representing the feedback taps. The constants d_i, $1 \leq i \leq L$, on the main diagonal represent feedback cells, and here half of the d_i's are equal to 0 and the other half are equal to 1. If there is only one feedback tap at position n, then all t_i's are 0, except for t_n. From this transition matrix the characteristic polynomial can be determined directly:

$$C(x) = 1 + \sum_{i=0}^{L-1} t_i \prod_{j=i+1}^{L} (d_j + x) \ ,$$

where $t_0 = 1$ is introduced for simplicity of the formula. Taking the aforementioned restrictions on the d_i's into account, and assuming only t_n is non-zero with k F-cells in the first n cells, one arrives at:

$$C_{n,k}(x) = x^{\frac{L}{2}}(1+x)^{\frac{L}{2}} + x^{\frac{L}{2}+k-n}(1+x)^{\frac{L}{2}-k} + 1 \ . \tag{2}$$

The tap of the trinomial and the positions of ones on the main diagonal are chosen in such a way that the characteristic polynomial is primitive and is neither self-reciprocal nor self-dual nor dual-reciprocal, i.e., they belong to a primitive S_6 set, that is a set of six primitive polynomials which are each others reciprocals and duals (for the details see [3]). Jump indices of the polynomials in S_6 set are coprime with the period λ. In particular, this means that the jump index of the characteristic polynomial satisfies $\gcd(J-1, \lambda) = 1$. The latter property is needed to provide the maximal period, that will be discussed further in Section 5. Choosing A to be of such a form we guarantee that the same number of XORs are used irrespective of the jump control signal that defines whether the LFSM is stepped once or makes a jump.

3 Description of the CJCSG

The CJCSG is the binary one clock pulse cascade clock control sequence generator with a bit stream output that operates in the Initialization Value (IV) Accommodation Mode. The key size is 128 bits and the IV length is allowed arbitrary in the range from 64 to 112 bits. The CJCSG consists of eight identical sections plus the incomplete ninth section that has the Jump Register (JR) only.

Section Keys. The 128-bit key K is split into eight 16-bit section keys that will be denoted as K_i ($i = 1, \ldots, 8$). The most significant bit (msb) of K is the msb of K_1, and so on, the least significant bit (lsb) of K is the lsb of K_8.

Jump Register. The JR implements a Linear Finite State Machine (LFSM) built on 14 memory cells. As shown in Fig. 1, cells can behave either as simple delay shift cells (S-cells) or feedback cells (F-cells) depending on the value of the Jump Control (JC) signal. Due to this mechanism, the diagonal entries in the transition matrix (1) of the LFSM are inverted, thereby creating the jump behavior. Both the number of S-cells and the number of F-cells in the JR is

equal to 7. This means that for both values of the JC bit there are 7 S-cells and 7 F-cells in the JR. Fig. 2 shows the configuration of cells that corresponds to the zero value of the JC. When JC is one then all the cells are switched to the opposite mode. The JR is a feedback shift register with a trinomial characteristic polynomial having the tap position at cell 6. For these values of $n = 6$ and $k = 2$ the characteristic polynomial of the LFSM (see (2)) is primitive with the jump index 5945.

Key Map. The 9-bit input vectors for the Key Map are composed of the cells numbered 2, 3, 4, 5, 7, 8, 9, 10, 11 of the Jump Register and are considered as the numbers (denoted as v) in the range from 0 to $2^9 - 1$ with the bit from cell 2 being the least significant and from cell 11 the most significant in v. Next, 9 least significant bits of the section key are bitwise XORed to v with the lsb of v XORed with the lsb of the section key. The sum (considered as a 9-bit number) is substituted by the 9-to-7 bit S-box which lookup table is provided in Appendix A. The result (denoted as w) is taken as a 7-bit vector and is bitwise XORed to the 7 most significant bits of the section key with the msb of w XORed with the msb of the section key. The resulting 7-bit sum is considered as a number and is fed into the Boolean function F which lookup table is provided in Appendix A. The output of F is called the "JC out" bit of the section and denoted as $\mathrm{JC_o}$.

Jump Register Section. A complete jump register section is shown in Fig. 2. It consists of the Jump Register and the Key Map. The Key Map implements a key-dependent filter function on the state of the JR and contains a 9-to-7 bit S-box and a balanced nonlinear Boolean function of 7 variables. In the Key Stream Generation mode (see Fig. 3) Jump Control bit (called "JC in" and denoted $\mathrm{JC_i}$) for section 1 is constantly 0. JC in for section i with $i \in \{2, \ldots, 9\}$ is the sum of the $\mathrm{JC_o}$ and $\mathrm{JC_i}$ of section $i - 1$. Section 9 consists of the JR only and does not have the Key Map. Denote the jump register in section i as JR$-i$.

Key Stream Generation Mode. Key stream is produces as an XOR sum of the taps from all 9 registers. The tap is taken from the cell 13 of the jump registers.

Shift Mode. This mode is used only during the initialization of the CJCSG (see Fig. 4 and Fig. 2). In this mode the $\mathrm{JC_o}$ (the Key Map output) of section i ($i = 1, \ldots, 8$) is added to the feedback of the JR$-(i + 1)$. The tap from cell 1 in the JR-9 is added to the feedback of the JR-1 and this closes "the big loop". Configuration of the jump registers does not change in the Shift Mode, they all operate as if the JC bit was constantly zero. In a software implementation the repeated sequence of steps in the Shift Mode is following: save the contents of the cell 1 in the JR-9; calculate $\mathrm{JC_o}$ for all the sections from 1 to 8; update the state of all the sections from 1 to 9.

Initialization of the CJCSG. The CJCSG needs to be initialized before starting the key stream generation. Firstly, preset the state of the jump register i ($i = 1, \ldots, 9$) to the value of pi[i] with the lsb of pi[i] coming in cell 1 of the register. Then run the generator for 128 steps in the Shift Mode. Finally, save

the 14-bit state of all 9 jump registers (call it the Initialization Vector) for the later use in the IV Mode.

IV Mode. Firstly, load all 9 jump registers with the Initialization Vector that was saved earlier during the initialization phase. IV can have arbitrary length in the range from 64 to 112 bits. Further, XOR the IV bitwise to the contents of the jump registers in such a way that 14 most significant bits of the IV are XORed with the JR-1 (msb of the IV is XORed with the msb of JR-1). The next 14 bits of the IV are XORed similarly to JR-2 and so on till we run out of the IV bits. When this happens we proceed cyclically starting again from the msb of the IV. The runup of the generator consists of 128 steps in the Key Stream Generation Mode but output bits unused.

After the runup the CJCSG starts generating the key stream in the Key Stream Generation Mode. Initialization of the CJCSG is done only once for a given key. Therefore, using the Initialization Vector allows to achieve fast start of the new IV session and re-synchronization.

4 Implementation

Hardware. The CJCSG is ideally suited for implementation in hardware. It can be implemented using standard components and has no complex circuits causing timing bottlenecks. The linear shift register part (Jump Register) uses 14 memory cells, each with an EXOR and switch. Typically, this takes about 170 gates (two-input equivalent). The 9-to-7 S-box in the Key Map is the most expensive real-estate, followed by the 7-to-1 Boolean function and the 16 EXORs. Implementation of these components by direct synthesis of the Boolean circuitry is estimated at 1000 gates. No attempts have been made to optimize the footprint of these circuits, by any means. For the complete design a total estimate is obtained of $8 \cdot 1000 + 9 \cdot 170 \approx 9500$ gates. A reduction of the gate-complexity of the S-box could lower this number substantially.

Software. In a software implementation of the CJCSG we need 512 bytes of data memory for the S-box lookup table plus the storage for the 7-to-1 Boolean function that can be reduced just to 16 bytes if the bits are packed into the byte. In total we can do with about 600 bytes for data plus something for the code. In a really compact implementation (although, much slower) we can replace the table lookup for the S-box with the algebraic calculation of the multiplicative inverse in the finite field. On the other hand, in the fastest implementation we can make a precalculation for 8 Key Maps (they depend on the key) and save them for the lookup during the keystream generation. Every Key Map is a 9-to-1 Boolean function and keeping 8 lookup tables will cost 512 bytes of memory. Our straightforward implementation of the CJCSG using portable C and Microsoft Visual Studio .NET 2003 compiler (no Key Map precalculation was done) without any code optimization gave the speed of 16 Mbits per second on Pentium 4, 2.8 GHz with 1GB RAM. Optimization of the code will considerably improve the speed. Moreover, the CJCSG is easy to parallelize, the property that can be used on some platforms.

5 Period and Linear Complexity

The CJCSG consists of N section of the similar type. We will number the sections from 1 to N starting with the rightmost section that is clocked regularly. Consider section number $i > 1$ of the CJCSG. It consists of the LFSR of length L which clocking is controlled by the binary Jump Control (JC) signal. Zero value in the JC signal makes the LFSR shift c_0 times and one makes it shift c_1 times. Assume that the JC sequence cycles periodically with the period $\pi_i = \lambda^{i-1}$ where $\lambda = 2^L - 1$ and there are N_i^0 zeroes and N_i^1 ones in the period. Obviously, $N_i^0 + N_i^1 = \lambda^{i-1}$. Denote $S_i = c_0 N_i^0 + c_1 N_i^1$ that is equal to the total number of shifts the LFSR makes when the JC sequence runs over its full period. Assume also that the characteristic polynomial of the LFSR is primitive of degree L and order λ.

Consider the sequence of LFSR states obtained when the clocking is controlled by the JC sequence and denote this sequence of states as u that is further called the output. We assume that the initial LFSR state is nonzero which means that the zero state will never be found in the output sequence. It is known (see, for instance, [5, Chapter 3] and [6]) that the period of the output sequence divides $\frac{\pi_i \lambda}{\gcd(S_i, \lambda)}$ and from [7, Lemma 1] it also follows that this period is a multiple of $\frac{\pi_i' \lambda}{\gcd(S_i, \lambda)}$ where π_i' is the product of all prime factors of π_i, not necessarily distinct, which are also factors of $\frac{\lambda}{\gcd(S_i, \lambda)}$. In particular, if every prime factor of π_i also divides $\frac{\lambda}{\gcd(S_i, \lambda)}$ then the period of u reaches the maximal value $\frac{\pi_i \lambda}{\gcd(S_i, \lambda)}$. This will be the case if we provide $\gcd(S_i, \lambda) = 1$.

Now for $i > 1$ consider the $\gcd(S_i, \lambda)$ with

$$S_i = c_0 N_i^0 + c_1 N_i^1 = c_0 (N_i^0 + N_i^1) + (c_1 - c_0) N_i^1 = c_0 \lambda^{i-1} + (c_1 - c_0) N_i^1 \ .$$

By the appropriate selection of the jump indices we guarantee that $\gcd(c_1 - c_0, \lambda) = 1$ (in our case one of the c_i is 1 and the other is J or $J^{\perp}$). Then $\gcd(S_i, \lambda) = \gcd((c_1 - c_0) N_i^1, \lambda) = \gcd(N_i^1, \lambda)$. Recall that the JC sequence is obtained as a sum of the Key Map output from the previous section and the JC signal for the previous section. Exception is the second section where the JC sequence is just the Key Map output from the first section.

Further we apply induction on $i > 1$ to prove that $\gcd(S_i, \lambda) = 1$. For $i = 2$ (the induction base) the JC sequence of the second section is the Key Map output from the first section that is a filtered m-sequence of period λ. Since the filter function (the Key Map) is balanced, then N_2^1 is either equal to 2^{L-1} or $2^{L-1} - 1$ depending on the value the filter function takes on the all-zero input vector. Thus, $\gcd(S_2, \lambda) = \gcd(N_2^1, \lambda) = 1$. Now assume that $\gcd(S_i, \lambda) = \gcd(N_i^1, \lambda) = 1$.

It is easy to see that any uniform π_i-decimation of the output sequence u is a uniform S_i-decimation of the original LFSR sequence of states. If $\gcd(S_i, \lambda) = 1$ then the latter decimation has period λ and contains all the nonzero states of the LFSR. We can write down sequence u row-by-row in a matrix with π_i columns and λ rows that will contain the full period of u. Each column of the matrix contains all the nonzero states of the LFSR. Let ν denote the number of nonzero

states of the LFSR producing a one when fed into the Key Map of section number i. Since the Key Map is a balanced Boolean function, then ν is either equal to 2^{L-1} or $2^{L-1}-1$ depending on the value the filter function takes on the all-zero input vector. We can write down the JC sequence of period π_i that controls the section number i in another matrix of the same size. This matrix will consist of N_i^1 columns containing only ones and $N_i^0 = \pi_i - N_i^1$ columns containing only zeros. Adding the matrices we get the full period of the JC sequence for the next section with

$$N_{i+1}^1 = (\lambda - \nu)N_i^1 + \nu(\pi_i - N_i^1) = \lambda N_i^1 + \nu\lambda^{i-1} - 2\nu N_i^1$$

and

$$\gcd(S_{i+1}, \lambda) = \gcd(N_{i+1}^1, \lambda) = \gcd(2\nu N_i^1, \lambda) = \gcd(N_i^1, \lambda) = 1$$

by the induction hypothesis.

Therefore, provided primitive characteristic polynomials for all the sections of the CJCSG, section number i generates the output sequence of the maximal period λ^i. Note that if just the Key Map output from the previous section was used to control the clocking then we would have

$$\gcd(S_{i+1}, \lambda) = \gcd(N_{i+1}^1, \lambda) = \gcd(\nu\lambda^{i-1}, \lambda) = \lambda \neq 1$$

for $i > 1$.

On the other hand, using [8, Theorem 2] we can evaluate the linear complexity of the component sequences of the output u. In particular, if the LFSR characteristic polynomial is primitive and $\gcd(S_i, \lambda) = 1$ then any component sequence taken from the output of the section number i is a linear recurring sequence with irreducible characteristic polynomial of degree $\lambda^{i-1}L$ giving the maximal linear complexity. N component sequences taken from the output of each section are XORed to produce the key stream. Characteristic polynomials of these component sequences are irreducible and have different degrees $\lambda^{i-1}L$ for $i = 1, \ldots, N$ which means that they are pairwise coprime. Thus, by [9, Theorem 8.57], the linear complexity of the key stream sequence is equal to $L(1+\lambda+\lambda^2+\ldots+\lambda^{N-1})$ and the period is equal to λ^N.

Note that every component sequence taken from the output of the section number i contains $\lambda^{i-1}(2^{L-1}-1)$ zeros and $\lambda^{i-1}2^{L-1}$ ones in the period. XOR of N sequences allows to compensate for this imbalance.

6 Security Analysis of the Cipher

The most important aspect of a cipher security is its resistance to different attacks. The goal is make any attack at least as difficult as the exhaustive search. Consider some general attacks on stream ciphers. We always assume the known plain text scenario when the attacker knows the key stream. **No weak keys have been identified.**

Exhaustive Key Search. This is the most efficient attack against the CJCSG. Searching through the whole key space gives the complexity of 2^{128}.

Time-Memory Trade-off. Assume that the attacker knows the state of the jump registers right before the generator starts producing the key stream. Then the kind of meet in the middle attack can be launched. The procedure is as follows. Take all possible 2^{16} keys that define the Key Map of section 8 (denote it K) and take all 2^n binary sequences of length n as the jump control for section 8 (denote this $\boldsymbol{a}$). For each combination generate the sequence of length n that is the key stream contribution from section 9 (denote it $F(K, \boldsymbol{a})$). Put the vector $(F(K, \boldsymbol{a}), \boldsymbol{a}, K)$ in a list sorted along $(F(K, \boldsymbol{a}), \boldsymbol{a})$. The value of n is chosen to be minimal with the property that the multi-set

$$\{(F(K, \boldsymbol{a}), \boldsymbol{a}) \mid K \in V_{2^{16}}, \boldsymbol{a} \in V_{2^n}\}$$

consists of different vectors. Then obviously, $n \geq 16$ and assuming the randomness of the F mapping we can take $n = 16$.

Run the exhaustive search on the remaining $128 - 16 = 112$ bits of the key. Calculate the sum of the key stream contributions from sections 1 to 8, add it to the key stream (get n bits like that) and also calculate n bits of the jump control sequence for section 8. If n is taken to be equal 16 then for each choice of the remaining 112 bits of the key we will find one match in the pre-computed list. The final elimination of wrong keys is done by generating and matching more bits in the jump control sequence for section 8 and the key stream contribution from section 9.

The total computational complexity consists of $O(2^{16+n})$ in pre-computation plus $O(2^{112})$ in the main phase. The lowest time complexity of the attack is achieved if we start with trying 32 bits of the key (take the last 2 sections and not just one). Then we need $O(2^{32+n})$ bits of memory and the computational complexity is $O(2^{32+n})$ in pre-computation plus $O(2^{96})$ in the main phase. If n is equal 32 then the total complexity will have the order of $O(2^{96})$. It can be concluded that if the internal state of the generator just before it starts producing the key stream is made secret then security against this type of the attacks is achieved.

Timing, Power and Side-Channel attacks. Resistance against timing attacks is inherent of the CJCSG and is achieved due to the use of jump control instead of the traditional clock control. Power and side-channel attacks are additionally countered by the important feature that the same number of XORs are used in each section of the generator irrespective of the jump control signal.

Fault Analysis Attacks. These attacks are countered due to the nonlinear functions in conditional jumping, accumulation of JC signals and accumulation of key stream outputs from individual LFSMs.

Guess and Determine Attacks. We have not found any attack of this kind on the proposed cipher.

Distinguishing Attacks. The distinguishing attack is assumed to succeed if the attacker can distinguish the key stream from the purely random sequence. It is reasonable to assume that the needed key stream length does not exceed the total number of keys for the generator since the distinguishing attack should not run longer than the exhaustive key search. The key stream produced by the

CJCSG is obtained as a sum of linear recurring sequences and this makes any statistical weaknesses in the key stream unlikely. The alternative is to look for the regularities during the initialization phase but we were not able to find any of this kind.

Another approach would be to consider a set of key stream sequences generated with the same key but for different IV values trying to find some dependencies between them that can not be found in the set of random independent sequences. This is also related to differential attack considered next.

Differential Attacks. This type of attacks, that was initially introduced for block ciphers, can also be applied to stream ciphers (see [10]). For synchronous stream ciphers differential attacks can use the known difference in the IV value. Usually is assumed that the attacked can choose the IV. The important precaution against these attacks against CJCSG is limiting the IV length to 112 bits and cyclical filling of the IV into the jump registers. This makes impossible (by choosing a suitable pair of IVs) to provide that the state, the jump registers get after loading these IVs, differ just in the last 9th section. Having two key stream sequences generated from the states that differ only in the 9th register the attacker can recover the jump control sequence for section 9. We will consider this attack in more details later in the extended version of this paper.

References

1. Jansen, C.J.: Modern stream cipher design: A new view on multiple clocking and irreducible polynomials. In González, S., Martínez, C., eds.: Actas de la VII Reunión Española sobre Criptología y Seguridad de la Información. Volume Tomo I. Servicio de Publicaciones de la Universidad de Oviedo (2002) 11–29
2. Jansen, C.J.: Partitions of polynomials: Stream ciphers based on jumping shift registers. In: 26th Symposium on Information Theory in the Benelux, Enschede, Werkgemeenschap voor Informatie- en Communicatietheorie (2005)
3. Jansen, C.J.: Modern streamcipher design: On multiple clocking and irreducible polynomials (2003)
4. Jansen, C.J.: Streamcipher design: Make your LFSRs jump! In: The State of the Art of Stream Ciphers, Workshop Record, ECRYPT Network of Excellence in Cryptology (2004) 94–108
5. Kholosha, A.: Investigations in the Design and Analysis of Key-Stream Generators. PhD thesis, Technische Universiteit Eindhoven (2003)
6. Kholosha, A.: Clock-controlled shift registers and generalized Geffe key-stream generator. In Rangan, C.P., Ding, C., eds.: Progress in Cryptology - INDOCRYPT 2001. Volume 2247 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2001) 287–296
7. Golić, J.D.: Periods of interleaved and nonuniformly decimated sequences. IEEE Transactions on Information Theory **44** (1998) 1257–1260
8. Chambers, W.G.: Clock-controlled shift registers in binary sequence generators. IEE Proceedings - Computers and Digital Techniques **135** (1988) 17–24
9. Lidl, R., Niederreiter, H.: Finite Fields. Volume 20 of Encyclopedia of Mathematics and its Applications. Addison-Wesley, Amsterdam (1983)

10. Muller, F.: Differential attacks and stream ciphers. In: The State of the Art of Stream Ciphers, Workshop Record, ECRYPT Network of Excellence in Cryptology (2004) 133–146

A S-Box and Function for the Key Map

S-box is defined by the inversion operation in the multiplicative group of $\mathrm{GF}(2^9)$ when the the finite field is defined by the primitive polynomial $f(x) = x^9 + x + 1$.

```
unsigned char S[512] = {
0,0,0,127,64,85,127,54,96,18,42,57,63,83,91,51,112,17,73,38,21,
103,92,49,95,122,105,113,45,104,25,61,120,107,8,112,100,89,19,39,
74,102,115,41,110,80,88,119,47,62,61,15,52,29,56,88,22,16,52,26,
12,125,94,93,124,75,53,14,4,77,120,84,114,2,44,112,73,9,19,19,
101,121,115,21,57,5,20,115,55,72,104,14,108,63,59,116,87,121,31,
89,94,80,7,91,90,98,14,33,92,84,44,72,75,82,72,82,90,85,13,48,70,
97,62,34,47,24,46,108,126,91,101,76,26,69,71,119,66,30,38,95,60,
97,106,117,57,82,65,78,86,78,56,82,100,111,4,34,73,65,9,51,50,94,
124,87,57,72,10,77,92,54,2,64,74,78,121,48,27,56,100,18,52,98,7,
51,54,84,31,94,93,31,122,12,43,29,60,70,79,5,108,110,111,76,40,
121,3,39,45,68,45,14,113,13,71,117,16,120,46,63,42,1,22,80,100,
76,37,44,105,13,36,2,41,21,109,125,106,71,70,122,88,23,35,84,48,
87,95,12,81,7,87,81,12,30,23,105,54,3,127,1,109,42,114,36,102,39,
77,34,98,79,99,117,123,81,97,86,79,51,83,77,111,33,30,125,48,59,
53,33,58,123,28,22,41,27,96,4,39,19,43,115,103,10,28,16,105,126,
50,114,55,32,66,69,17,41,36,37,96,43,68,66,89,49,25,55,111,11,62,
61,107,67,28,37,36,28,69,95,102,3,46,60,27,17,1,109,96,29,37,112,
103,68,60,40,24,62,13,59,92,11,114,24,9,79,26,29,113,106,3,127,25,
32,27,88,42,5,15,123,47,116,46,40,15,25,61,34,6,83,85,2,78,73,30,
68,35,107,103,45,66,26,118,122,119,67,55,44,38,9,20,102,124,32,65,
101,83,10,86,74,98,5,22,110,7,123,56,75,6,63,35,120,58,90,8,97,
124,81,23,119,31,49,85,58,64,126,11,49,104,118,50,80,38,69,18,4,
86,8,52,90,6,117,18,89,65,76,20,74,10,21,118,93,126,23,53,113,35,
67,99,110,125,116,108,99,11,33,17,8,106,53,24,50,43,20,47,59,6,99,
104,93,67,71,107,16,40,101,70,118,15,58,75,32,116,109,91,64,1,0};
```

Boolean function F of 7 variables is 2-resilient of degree 4 and nonlinearity 56.

```
unsigned char F[128] = {
0,1,1,1,1,0,0,1,0,1,1,0,1,0,0,1,1,0,0,0,0,0,0,1,0,1,1,1,1,1,0,0,
1,1,0,0,0,1,0,1,1,0,0,0,1,0,0,1,0,0,1,1,1,0,1,1,1,0,1,0,0,1,1,0,
1,0,1,0,1,1,0,0,0,0,1,1,0,0,1,0,0,1,1,0,1,1,1,0,0,1,0,0,0,1,1,1,
0,1,1,0,0,0,0,1,1,0,0,1,1,1,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,0,0,0};
```

Initial state of the jump registers.

```
unsigned short pi[9] = {
0x90F, 0x36A8, 0x2216, 0x2308, 0x34C4,
0x3198, 0x28B8, 0x370, 0x1CD1};
```