Cascade Jump Controlled Sequence Generator and Pomaranch Stream Cipher

(Version 3)

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

[1] DeltaCrypto BV
Jv. Riebeeckstr. 10
5684 EJ Best, The Netherlands
[2] The Selmer Center
Department of Informatics, University of Bergen
P.O. Box 7800, N-5020 Bergen, Norway
cja@iae.nl; {Tor.Helleseth,Alexander.Kholosha}@uib.no

Abstract. Jump registers were recently proposed [SASC04] as building blocks for stream ciphers. In this paper a construction based on these principles is described. The proposed encryption primitive is a synchronous stream cipher accommodating a key of 128 bits and an IV of 64 up to 162 bits, or an 80-bit key and 32 to 108 bit IV. Version 3 comes as a final submission for the second phase of the eSTREAM project. The stream cipher is particularly designed to resist side-channel attacks and can be efficiently implemented in hardware for a wide range of target processes and platforms.

Key words: stream cipher, Pomaranch, jump register.

1 Introduction

Linear feedback shift registers (LFSR's) are known to allow fast implementation and produce sequences with a 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 an LFSR through successive states. Key-stream generators based on regularly clocked LFSR's are susceptible to basic and fast correlation attacks. Use of irregular clocking limits the possibilities for mounting classical correlation attacks.

Due to the multiple clocking, key-stream generators that use clock-controlled LFSR's 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, power and other side-channel 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 registers 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.

Pomaranch is a stream cipher that follows a classical design of synchronous bit-oriented stream ciphers and consists of a key-stream generator producing a secure sequence of bits that is further XORed with the plain text previously converted into bits. The key-stream generator of Pomaranch is called Cascade Jump Controlled Sequence Generator (CJCSG) and is primarily intended for hardware implementation. Along with providing an appropriate security level it can be used in a wide range of hardware platforms included those having very limited computing and memory resources (see Section 4). However, our current generator can hardly reach the bit generation rate achieved by word-oriented algorithms especially designed for software implementation. Therefore, the software use of the bit-oriented CJCSG is mostly interesting from the academic point of view. 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 by now (see [2, 3]). This will be implemented in future versions of the CJCSG.

Following is the list for tracking the changes introduced in Version 2 (see [4]) of the CJCSG compared to the original version [5]. Last three of these changes came as a countermeasure against key-recovery attacks found for the original Pomaranch in [6–8].

1. Hardware-oriented 80-bit key version of the CJCSG is added. The only difference between the full 128-bit version and the 80-bit version is the total number of jump register sections that is equal respectively to 9 and 6 and the number of Shift Mode steps during the IV setup that is equal to 96 and 80 respectively.
2. Feedback taps of jump registers are taken now from cells number 4, 8 and 14. The positions of the F- and S-cells in the registers are FFSFFFSSFSSFSS.
3. Input to the Key Map is taken from the cells of the jump registers number 1, 2, 3, 5, 6, 7, 9, 10, 11.
4. The new IV setup procedure is described in Section 3 under the subtitle "IV Setup".

The need for a new Version 3 was brought up by the attacks found in [9]. The changes introduced compared to Version 2 [4] are as follows.

1. Instead of one type of jump register consisting of 14 bits, two different types of jump registers of 18 bits are used. Type 1 is used for the odd numbered sections and type 2 for the even numbered sections.

2. The feedback taps of the type 1 jump registers are taken from cells number 3, 8, 16 and 18. The type 2 jump registers have feedback taps at cells 6, 8, 14 and 18. The positions of the F- and S-cells in the type 1 registers are FSFFFSFFFFSSSSSFSS (see Fig. 2). The type 2 F- and S-cell positions are SSFSFFFFFFSFSSSFSS (see Fig. 3).
3. Input to the Key Map of the type 1 jump registers is taken from cells 1, 2, 4, 5, 6, 7, 9, 10, 11 (see Fig. 2). Input to the Key Map of the type 2 jump registers is taken from cells 1, 2, 3, 4, 5, 7, 9, 10, 11 (see Fig. 3).
4. The key-stream contribution is taken from the cell 17 of the jump registers.
5. In the 80-bit version, the XOR function on the outputs from jump register sections 1 to 5 is replaced with the nonlinear function G which output is XORed to the contribution from section 6 (see Fig. 4 and Appendix A).
6. The maximum IV length has increased to 162 bits for the 128-bit key version and to 108 bits for the 80-bit key version.
7. The number of Shift Mode steps during the IV setup has increased to 108 steps for the 128-bit key version and to 88 steps for the 80-bit key version.
8. The maximal length of the key-stream to be generated using one key-IV pair is limited to 2^{64} bits.

The Version 3 changes were motivated by the need to increase resistance against attacks involving linear relations in the output stream found in [9].

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 Håvard Raddum, Matthew G. Parker and Igor Semaev. 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 Registers

The ideas presented in this section are well described in [1, 10, 11, 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 a 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 time 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 [11, 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, namely,

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

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 contains constants t_i ($1 \le i \le L-1$), representing the feedback taps. Nonzero constants d_i ($1 \le i \le L$), on the main diagonal represent feedback cells, and half of the d_i's here are equal to 0 and the other half are equal to 1. The characteristic polynomial of this transition matrix 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_1}, t_{n_2} and t_{n_3} for $n_3 > n_2 > n_1$ are nonzero with k_1 feedback cells among cells 1 to n_1, k_2 feedback cells among cells n_1+1 to n_2 and k_3 feedback cells among cells n_2+1 to n_3 one arrives at

$$\begin{aligned} C(x) = 1 &+ x^{\frac{L}{2}+k_1+k_2+k_3-n_3}(x+1)^{\frac{L}{2}-k_1-k_2-k_3} + x^{\frac{L}{2}+k_1+k_2-n_2}(x+1)^{\frac{L}{2}-k_1-k_2} \\ &+ x^{\frac{L}{2}+k_1-n_1}(x+1)^{\frac{L}{2}-k_1} + x^{\frac{L}{2}}(x+1)^{\frac{L}{2}}. \end{aligned} \tag{2}$$

The feedback taps and the positions of ones on the main diagonal are chosen in such a way that the characteristic polynomial $C(x)$ is primitive and is neither self-reciprocal nor self-dual nor dual-reciprocal, i.e., it belongs 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 [11]). Jump indices of the polynomials in S_6 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 of the output sequence that will be discussed further in Section 5. Choosing A to be of such a form we guarantee that the same number of XOR's 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 a binary one clock pulse cascade clock control sequence generator with a bit stream output that operates in the Initialization Value (IV) accommodation mode. It is intended for hardware implementation and comes in two versions with 128-bit and 80-bit key length. These versions differ only in the number of jump register sections used and the number of Shift Mode steps during the IV setup. In the 128-bit version, the IV length is allowed arbitrary in the range from 64 to 162 bits. The 80-bit version accommodates the IV of 32 to 108 bits long. The 128-bit (80-bit) version of the CJCSG consists of eight (five) sections plus the incomplete ninth (sixth) section that has the Jump Register (JR) only. Hereafter and in the drawings attached at the end of the paper the total number of sections is denoted N, thus, either $N=9$ or $N=6$. The sections are numbered from 1 to N and every section having odd number is of type 1 (Fig. 2) and having even number is of type 2 (Fig. 3). Also denote the key length as κ. The initialization phase consists of key setup, IV setup and the run-up.

Section Keys. The κ-bit key K is split into $N-1$ 16-bit section keys that will be denoted as K_i $(i=1,\ldots,N-1)$. 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_{N-1}.

Jump Registers. There are two different types of Jump Registers (JR), differing in their cell configurations and feedback taps. A JR implements a Linear Finite State Machine (LFSM) built on 18 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 9. This means that for both values of the JC-bit there are 9 S-cells and 9 F-cells in the JR. Fig. 2 (resp. Fig. 3) shows the configuration of cells that corresponds to the zero value of the JC-bit for the odd (resp. even) numbered sections. When JC is one then all the cells are switched to the opposite mode. The JR of the odd numbered sections is a feedback shift register with a characteristic polynomial having the tap positions at cells 3, 8, 16 and 18. For these values of $n_1=3$, $n_2=8$, $n_3=16$ and $k_1=1$, $k_2=0$, $k_3=7$ the characteristic polynomial of the LFSM (see (2)) is primitive with the jump index 84074. The JR of the even numbered sections is a feedback shift register with a characteristic polynomial having the tap positions at cells 6, 8, 14 and 18. For these values of $n_1=6$, $n_2=8$, $n_3=14$ and $k_1=1$, $k_2=1$,

$k_3 = 6$ the characteristic polynomial of the LFSM (see (2)) is primitive with the jump index 27044.

Key Map. The 9-bit input vectors for the Key Map are composed of the cells numbered 1, 2, 4, 5, 6, 7, 9, 10, 11 of the type 1 jump register and 1, 2, 3, 4, 5, 7, 9, 10, 11 of the type 2 jump register. These 9-bit vectors are considered as the numbers (denoted as v) in the range from 0 to $2^9 - 1$ with the bit from cell 1 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. The two complete jump register sections are shown in Fig. 2 and Fig. 3. They consist 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. 4) Jump Control bit (called "JC in" and denoted $\mathrm{JC_i}$) for section 1 is constantly 0. JC in for section i with $i \in \{2, \ldots, N\}$ is the sum of the $\mathrm{JC_o}$ and $\mathrm{JC_i}$ of section $i - 1$. Section N consists of the JR only and does not have the Key Map. Denote the jump register in section i as R_i.

Key-Stream Generation Mode is shown in Fig. 4. In the 128-bit version, the key-stream is produced as an XOR sum of the taps from all 9 registers. In the 80-bit version, taps from the register sections 1 to 5 are combined and considered as a number providing an argument of the Boolean function G which lookup table is provided in Appendix A. Output of G is XORed with the tap from section 6. All the taps are taken from the cell 17 of the jump registers.

Shift Mode. This mode is used during the initialization and IV setup of the CJCSG (see Fig. 5 and Figs. 2,3). In this mode the $\mathrm{JC_o}$ (the Key Map output) of section i ($i = 1, \ldots, N-1$) is added to the feedback of the R_{i+1}. The tap from cell 1 in the R_N is added to the feedback of the R_1 and this closes "the big loop". The configuration of the jump registers does not change in the Shift Mode, they all operate as if the JC bit was constantly zero.

The Shift Mode is used to make the register contents depend on all initial content bits and all key bits. This mode defines a key dependent one-to-one mapping of the set of all $(18 \cdot N)$-bit states onto itself. Indeed, let $R_i^t = (r_{i,18}^t, \ldots, r_{i,1}^t)$ denote the 18-bit state of the register R_i ($1 \le i \le N$) and let $c_i^t = f_i(R_i^t)$ denote the output bit of the Key Map of section i ($1 \le i \le N-1$) at a time t. If A_i denotes the transition matrix (1) of register R_i which is fixed as if the JC bits were constantly zero, then the following equations define the Shift Mode:

$$R_1^{t+1} = R_1^t A_1 \oplus (0, \ldots, 0, r_{N,1}^t)$$

$$R_i^{t+1} = R_i^t A_i \oplus (0, \ldots, 0, f_{i-1}(R_{i-1}^t)) \quad (i = 2, \ldots, N)\ .$$

From the concrete form of matrices A_i applied in the Shift Mode it is clear that $r_{i,2}^{t+1} = r_{i,1}^t$ $(1 \le i \le N)$. So the inverse of the above equations can be written as

$$R_1^t = \left(R_1^{t+1} \oplus (0, \ldots, 0, r_{N,2}^{t+1})\right) A_1^{-1}$$
$$R_i^t = \left(R_i^{t+1} \oplus (0, \ldots, 0, f_{i-1}(R_{i-1}^t))\right) A_i^{-1} \quad (i = 2, \ldots, N)\ .$$

This shows that the Shift Mode defines an invertible onto mapping which needs to be a bijection.

Also note that in the Shift Mode the worst case diffusion of all IV bits is achieved after $N + 23 + 2 \cdot (N \bmod 2)$ steps, the respective number for IV-plus-key bits diffusion is $2N + 23 + 2 \cdot (N \bmod 2)$ steps.

Key Setup. Firstly, preset the state of the jump register i $(i = 1, \ldots, N)$ to the value of pi[i] (see Appendix A) 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 18-bit states of all N jump registers (call it the Initialization Vector) for later use during the IV setup.

IV Setup and the Run-up. The sequence of steps for the IV setup is the following:

1. The IV can have an arbitrary length in the range from 64 for the 128-bit version (32 in the 80-bit) to $18N$ bits. If the IV length is less than $18N$ then extend the IV to $18N$ bits by cyclically repeating its bits.
2. XOR the $18N$-bit (extended) IV with the Initialization Vector saved after the key setup and load the result into the N jump registers. The 18 most significant bits of the IV modify R_1 (msb of the IV modifies the msb of R_1), the next 18 bits of the IV similarly modify R_2 and so on.
3. Run the generator in the Shift Mode for $S = 108$ steps if $N = 9$ (128 key bits) or for $S = 88$ steps if $N = 6$ (80 key bits).
4. If any of the N registers has the all-zero state then set its least significant bit to 1.
5. Perform a run-up of 64 steps in the Key-Stream Generation Mode discarding the output bits.

After the run-up 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 a new IV session and re-synchronization. Since the Shift Mode defines a bijection, the suggested IV setup procedure not only guarantees a key dependent diffusion of the IV bits but also provides a different internal state before Step 4 for different IV's.

4 Implementation

Hardware. The CJCSG is ideally suited for hardware implementation since it requires standard components and has no complex circuits causing timing

bottlenecks. The 80-bit version of the CJCSG consists of 6 sections with 5 of them containing the Key Map. The linear shift register part (jump registers) uses 18 memory cells, each with an XOR and a switch. Typically, this takes about 225 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 16 XOR's. 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 means of a silicon compiler. For the complete design a total estimate is obtained of $5 \cdot 1000 + 6 \cdot 225 \approx 6300$ gates. Reduction of the gate-complexity of the S-box can lower this number substantially as can be seen from the following.

First note that the 9-to-7 S-box presented in Appendix A is defined by the inversion operation in the multiplicative group of GF(2^9) when the finite field is defined by the irreducible polynomial $f(x) = x^9 + x + 1$. Further the most and the least significant bits (msb and lsb) of the result are deleted to obtain a 7-bit value. We can define a more efficient (having lower gate-complexity) implementation of the inverse in GF(2^9) using inverses in the subfield GF(8), i.e., inverses are calculated in GF(8^3) instead. The elements of GF(8^3) are represented by polynomials of degree at most 2 over GF(8) and operations in the field are carried out modulo an irreducible polynomial $Q(x) = x^3 + a_2x^2 + a_1x + a_0$ over GF(8). Operations in GF(8) can be implemented with low complexity by table lookups using one of the following moduli $x^3 + x + 1$ or $x^3 + x^2 + 1$. Summing up all the above said, the following steps could lead to a lower complexity implementation of the S-box:

1. Find a primitive element of GF(2^9) modulo $x^9 + x + 1$ and calculate the polynomial $Q(x)$ (see [12]).
2. Let $b_2x^2+b_1x+b_0$ be the inverse modulo $Q(x)$ of a polynomial $c_2x^2+c_1x+c_0$ over GF(8). Find analytical expressions for the coefficients b_2, b_1, b_0 as a function of c_2, c_1, c_0 and a_2, a_1, a_0. These are found as a solution of a system of three linear equations in three unknowns that can be solved applying Cramer's rule. The operations required to calculate the b_i from the given c_i and a_i $(i = 0, 1, 2)$ are multiplications, additions and inverse in the subfield GF(8).
3. The number of subfield operations for finding the solutions amounts to 18 multiplications, 6 constant multiplications, 8 XOR's and 1 inverse.
4. The gate-complexity of multiplication and inverse in GF(8) is determined by finding the ANF's for the two irreducible polynomials and two bases each (Galois counter and LFSR basis). This results in: inverse between (6 gates and 1 inverter) and (10 gates and 3 inverters), where inverter means binary inverter, so say 10 gates; multiplication 17 or 18 gates; constant multiplication costs only 1 or 2 gates (XOR's). The total cost is therefore $18 \cdot 17 + 6 \cdot 2 + 8 + 10 = 336$ gates.
5. A linear transform and its inverse are needed to map 9-bit vectors to vectors over GF(8^3) and back, where the inverse transform is combined with the 7-to-1 Boolean function. The cost of these 9-by-9 matrices is estimated at

40 XOR's. Hence, the total cost is estimated at 400 gates (two-input AND, OR, XOR, etc).

We conclude that for a hardware implementation of 6 sections with $5 \cdot 16 = 80$ key bits the total gate-count would amount to $5 \cdot 400 + 6 \cdot 225 \approx 3300$ gates. Note two things here: in practice a good silicon compiler may even do better by reusing intermediate results at several places; the estimate for the gate-complexity needed to implement the full inverse while deleting the msb and lsb can further reduce the gate-count.

Software. The 128-bit version of the CJCSG consists of 9 sections. 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 bytes. 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 in a 512×8 bit table for lookup during the key-stream generation.

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 a speed of 16 Mbits per second on a 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.

A software implementation of an LFSR usually takes more instructions than an implementation of Galois counter registers. In the latter it suffices to test the msb of the register and conditionally add the feedback mask to the register. In an LFSR implementation the parity of the word containing the feedback tap bits needs to be determined, which takes one or more instructions extra, depending on the implementation and the platform used. As these extra instructions are to be executed for all N registers for every bit, this has a substantial impact on the performance of the software implementation. It should be clear that an equivalent CJCSG could be constructed using Galois counter registers with the appropriate feedback function and shift- and feedback cells by a similarity transform. This exercise, however, is beyond the scope of this document.

5 Period and Linear Complexity

The CJCSG consists of N sections. 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. A zero value in the JC signal makes the LFSR shift c_0 times and a 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, [13, Chapter 3] and [14]) that the period of the output sequence divides $\frac{\pi_i \lambda}{\gcd(S_i,\lambda)}$ and from [15, 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 the 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

$$\gcd(S_{i+1},\lambda) = \gcd(N^1_{i+1},\lambda) = \gcd(2\nu N^1_i,\lambda) = \gcd(N^1_i,\lambda) = 1$$

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

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

for $i > 1$.

On the other hand, using [16, 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. In the 128-bit version, $N = 9$ 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 $i = 1,\dots,N$ which means that they are pairwise coprime. Thus, by [17, Theorem 8.57], the linear complexity of the key-stream sequence is equal to $L(1+\lambda+\lambda^2+\ldots+\lambda^{N-1})$ and, by [17, Theorem 8.59], the period is equal to λ^N. In the 80-bit version, the maximal period is guaranteed by the XOR of the output from section $N = 6$ having period λ^N to the output from function G. The linear complexity is lower bounded by $\lambda^{N-1}L$.

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. The XOR (with nonlinear balanced function G for the 80-bit version) of output 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^κ with $\kappa = 128$ and $\kappa = 80$ for the two versions and corresponding key length. Like above, N denotes the total number of sections.

Correlation Attacks. A key-recovery attack [7, 18, 19] on the original Pomaranch was built due to the spotted biases in the distribution of certain linear relations of length $L+1$ in the output sequence of a jump register section. The suggested attack has the complexity $O(2^{87})$ and requires less than 2^{72} bits of

the key-stream. That became a primary reason for changing the configuration of jump registers in Version 2 of the cipher where it was guaranteed that no relation of length $L+1$ has a large enough bias. However, the updated configuration was also found to be insecure due to the new biased linear relation of a larger length found in [9]. Using this relation, a feasible key-recovery attack has the complexity $O(2^{94})$ requiring 2^{74} bits of the key-stream for the 128-bit key version and $O(2^{65})$ with 2^{45} bits for the 80-bit key version. Distinguishing attacks would have the same complexity but require less key-stream bits. By increasing the length of the registers to 18 and choosing new configuration we bring the bias of the best linear relation (that we were able to compute which is up to $L + 11 = 29$ bits long) for a separate register down to a level that brings the complexity of the attack up to the level exceeding the one of the exhaustive key search. Additionally, having jump register section of two different types and adding their outputs also decreases the resulting bias.

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 $N-1$ (denote it K) and take all 2^n binary sequences of length n as the jump control for section $N-1$ (denote this $\boldsymbol{a}$). For each combination generate the sequence of length n that is the key-stream contribution from section N (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 $\kappa - 16$ bits of the key. Calculate the sum of the key-stream contributions from sections 1 to $N-1$, add it to the key-stream (get n bits like that) and also calculate n bits of the jump control sequence for section $N-1$. If n is taken to be equal 16 then for each choice of the remaining $\kappa - 16$ 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 $N-1$ and the key-stream contribution from section N.

The total computational complexity consists of $O(2^{16+n})$ in pre-computation plus $O(2^{\kappa-16})$ in the main phase. If $\kappa = 128$ then 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 XOR's 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 LFSM's.

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. The distinguishing attack on Version 2 found in [9] is believed to be countered by a new configuration of the jump registers.

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 Chosen IV Attacks. This type of attacks, that was initially introduced for block ciphers, can also be applied to stream ciphers (see [20]). For synchronous stream ciphers differential attacks can use the known difference in the IV value. Moreover, usually it is assumed that the attacker can choose the IV. Two chosen IV key-recovery attacks on the original Pomaranch were recently found in [6, 8] and they exploit the weakness in the original IV setup procedure. The attack in [6] allows to recover the 128-bit key with the complexity $O(2^{65})$ or even faster, with $O(2^{52})$ if the escape from all-zero state feature in the initialization is used. The attack in [8] has a higher complexity of $O(2^{73.5})$ and is an extension of the correlation attack from [7]. This became the reason for introducing a new IV setup procedure in Version 2 of the cipher that provides good diffusion of IV bits. The updated versions are believed to be secure against this type of attacks.

References

1. Jansen, C.J.A.: 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.A.: Partitions of polynomials: Stream ciphers based on jumping shift registers. In Cardinal, J., Cerf, N., Delgrange, O., Markowitch, O., eds.: 26th Symposium on Information Theory in the Benelux, Enschede, Werkgemeenschap voor Informatie- en Communicatietheorie (2005) 277–284
3. Jansen, C.J.A.: Stream cipher constructions over binary extension fields. In Lagendijk, I., Weber, J., eds.: 27th Symposium on Information Theory in the Benelux, Enschede, Werkgemeenschap voor Informatie- en Communicatietheorie (2006) –

4. Jansen, C.J.A., Helleseth, T., Kholosha, A.: Cascade jump controlled sequence generator and Pomaranch stream cipher (Version 2). eSTREAM, ECRYPT Stream Cipher Project, Report 2006/006 (2006) http://www.ecrypt.eu.org/stream/papersdir/2006/006.pdf.
5. Jansen, C.J.A., Helleseth, T., Kholosha, A.: Cascade jump controlled sequence generator (CJCSG). In: Symmetric Key Encryption Workshop, Workshop Record, ECRYPT Network of Excellence in Cryptology (2005) http://www.ecrypt.eu.org/stream/ciphers/pomaranch/pomaranch.pdf.
6. Cid, C., Gilbert, H., Johansson, T.: Cryptanalysis of Pomaranch. In: SASC 2006, Stream Ciphers Revisited, Workshop Record, ECRYPT Network of Excellence in Cryptology (2006) 1–6 http://www.ecrypt.eu.org/stream/papersdir/060.pdf.
7. Khazaei, S.: Cryptanalysis of Pomaranch (CJCSG). eSTREAM, ECRYPT Stream Cipher Project, Report 2005/065 (2005) http://www.ecrypt.eu.org/stream/papersdir/065.pdf.
8. Hasanzadeh, M.M., Khazaei, S., Kholosha, A.: On IV setup of Pomaranch. In: SASC 2006, Stream Ciphers Revisited, Workshop Record, ECRYPT Network of Excellence in Cryptology (2006) 7–12 http://www.ecrypt.eu.org/stream/papersdir/082.pdf.
9. Hell, M., Johansson, T.: On the problem of finding linear approximations and cryptanalysis of Pomaranch version 2. to appear in SAC proceedings (2006)
10. Jansen, C.J.A.: 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 http://www.ecrypt.eu.org/stvl/sasc/sasc-record.zip.
11. Jansen, C.J.A.: Stream cipher design based on jumping finite state machines. Cryptology ePrint Archive, Report 2005/267 (2005) http://eprint.iacr.org/2005/267/.
12. Sunar, B., Savas, E., Çetin K. Koç: Constructing composite field representations for efficient conversion. IEEE Transactions on Computers **52**(11) (2003) 1391–1398
13. Kholosha, A.: Investigations in the Design and Analysis of Key-Stream Generators. PhD thesis, Technische Universiteit Eindhoven (2003) http://alexandria.tue.nl/extra2/200410591.pdf.
14. 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 LNCS., Berlin, Springer-Verlag (2001) 287–296
15. Golić, J.D.: Periods of interleaved and nonuniformly decimated sequences. IEEE Trans. Inform. Theory **44**(3) (1998) 1257–1260
16. Chambers, W.G.: Clock-controlled shift registers in binary sequence generators. IEE Proceedings - Computers and Digital Techniques **135**(1) (1988) 17–24
17. Lidl, R., Niederreiter, H.: Finite Fields. Volume 20 of Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge (1997)
18. Jansen, C.J.A., Kholosha, A.: Countering the correlation attack on Pomaranch. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/070 (2005) http://www.ecrypt.eu.org/stream/papersdir/070.pdf.
19. Helleseth, T., Jansen, C.J.A., Kholosha, A.: Pomaranch - design and analysis of a family of stream ciphers. In: SASC 2006, Stream Ciphers Revisited, Workshop Record, ECRYPT Network of Excellence in Cryptology (2006) 13–24 http://www.ecrypt.eu.org/stream/papersdir/2006/008.pdf.
20. 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 http://www.ecrypt.eu.org/stvl/sasc/sasc-record.zip.

A S-Box and Function for the Key Map

S-box is defined by the inversion operation in the multiplicative group of $GF(2^9)$ when the finite field is defined by the irreducible 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};
```

Boolean function G of 5 variables is 1-resilient of degree 3 and nonlinearity 12.

```
unsigned char G[32] =
{0,1,0,1,0,1,0,0,1,0,0,1,1,0,1,1,1,0,1,0,0,1,1,1,0,1,1,0,1,0,0,0};
```

Initial state of the jump registers.

```
unsigned long pi[9] = {
0x090FD, 0x2A888, 0x168C2, 0x0D313, 0x06628,
0x2E037, 0x01CD1, 0x0A409, 0x0E088};
```

Fig. 1 S-cells and F-cells

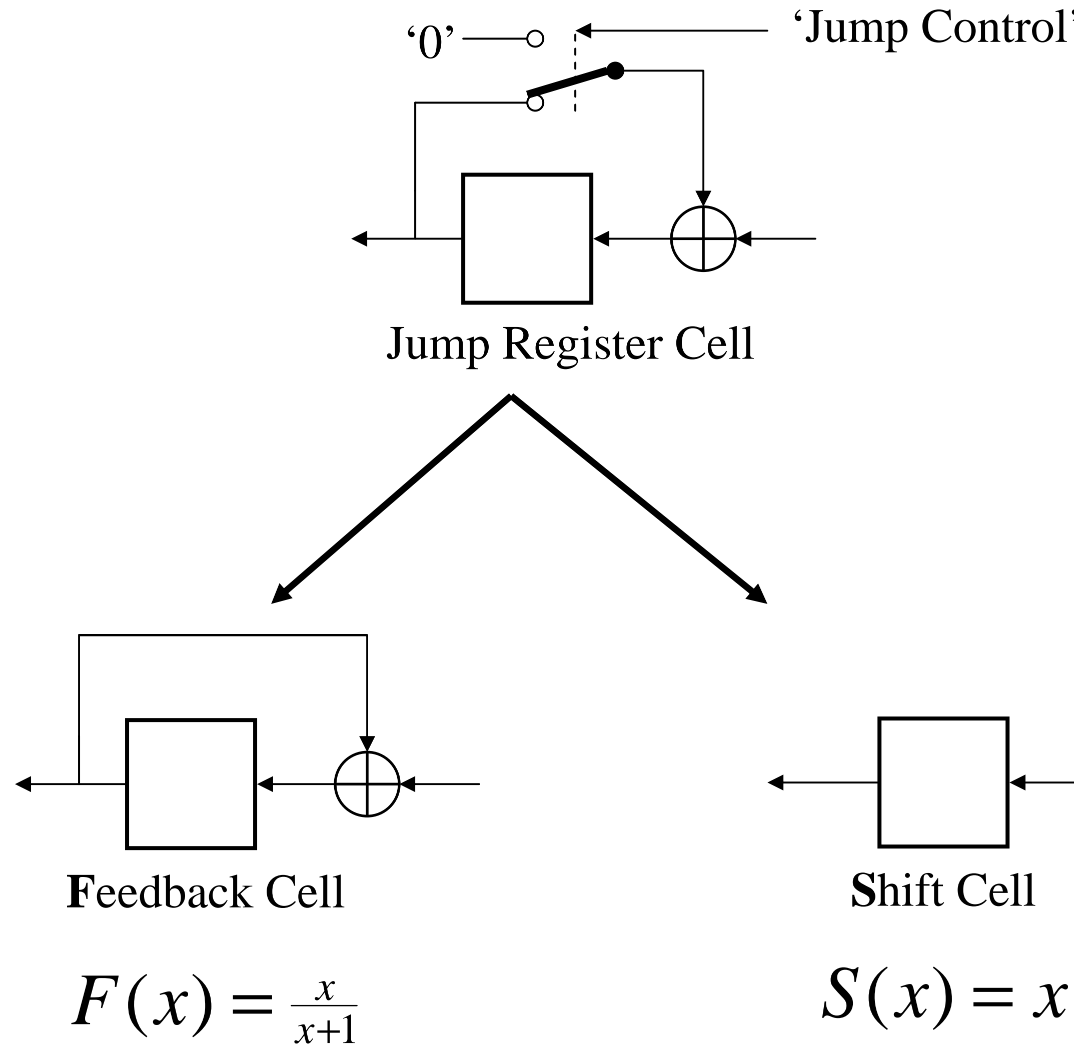


Fig. 2 Jump Register Section type 1

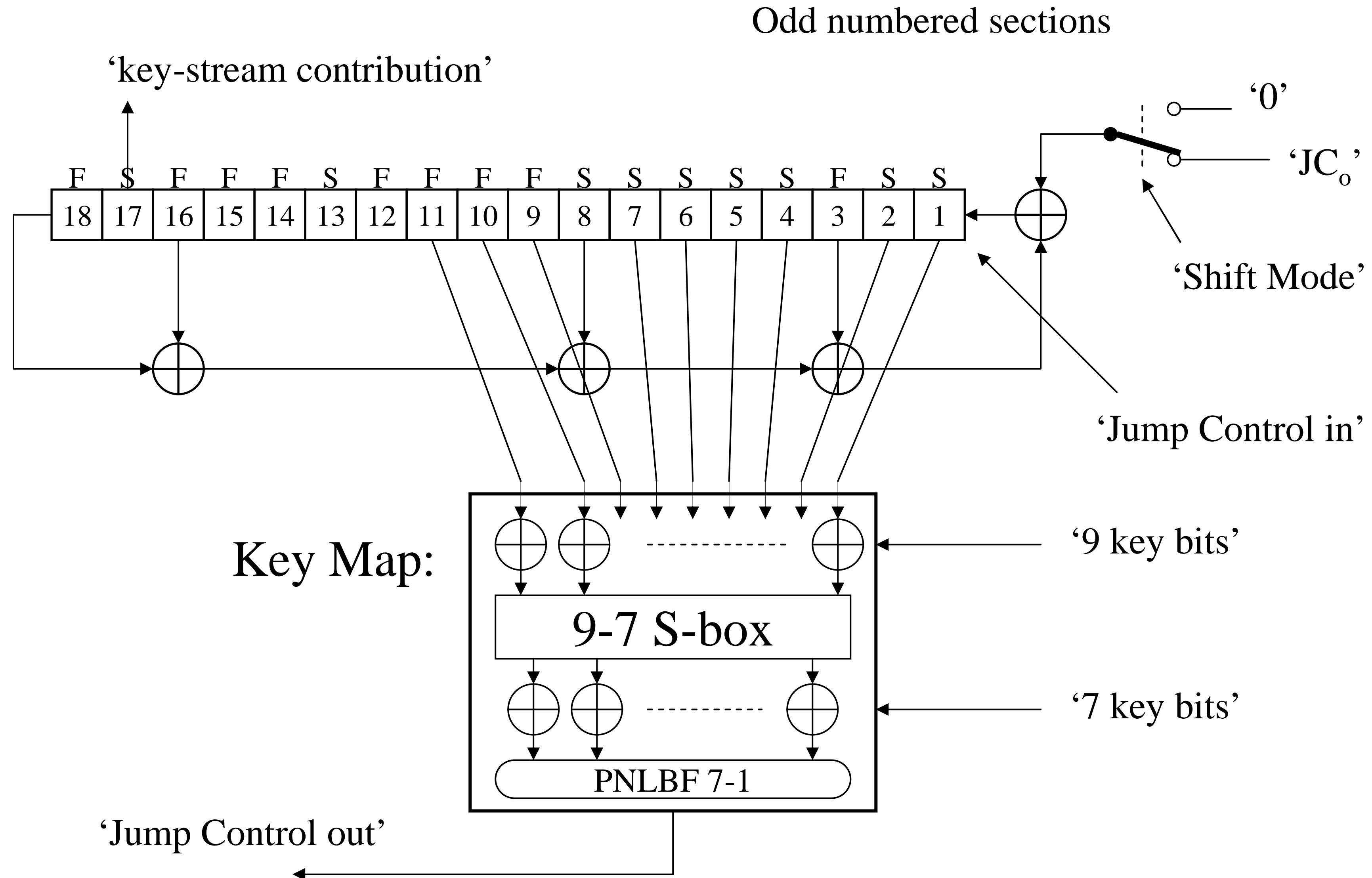


Fig. 3 Jump Register Section type 2

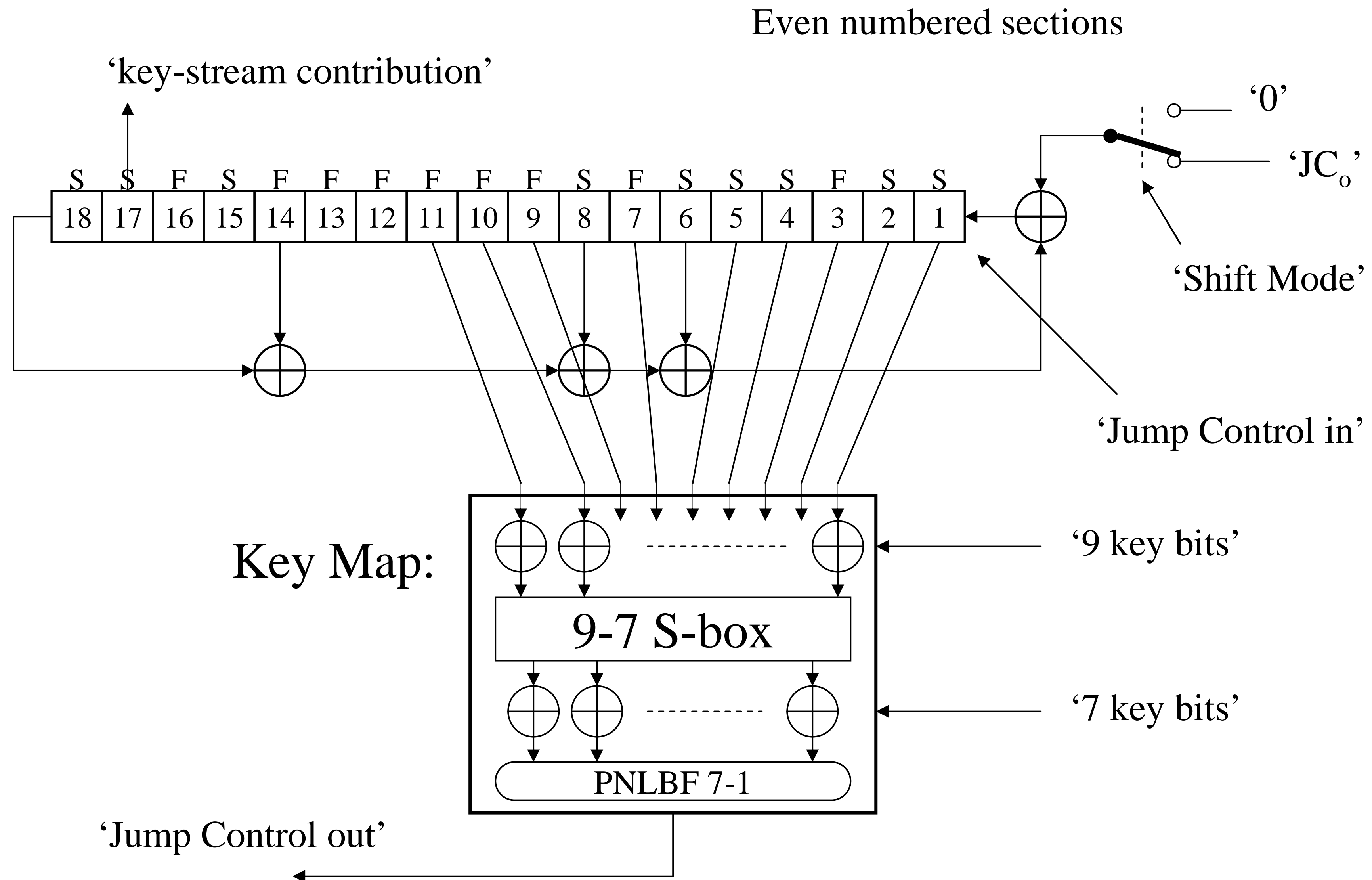
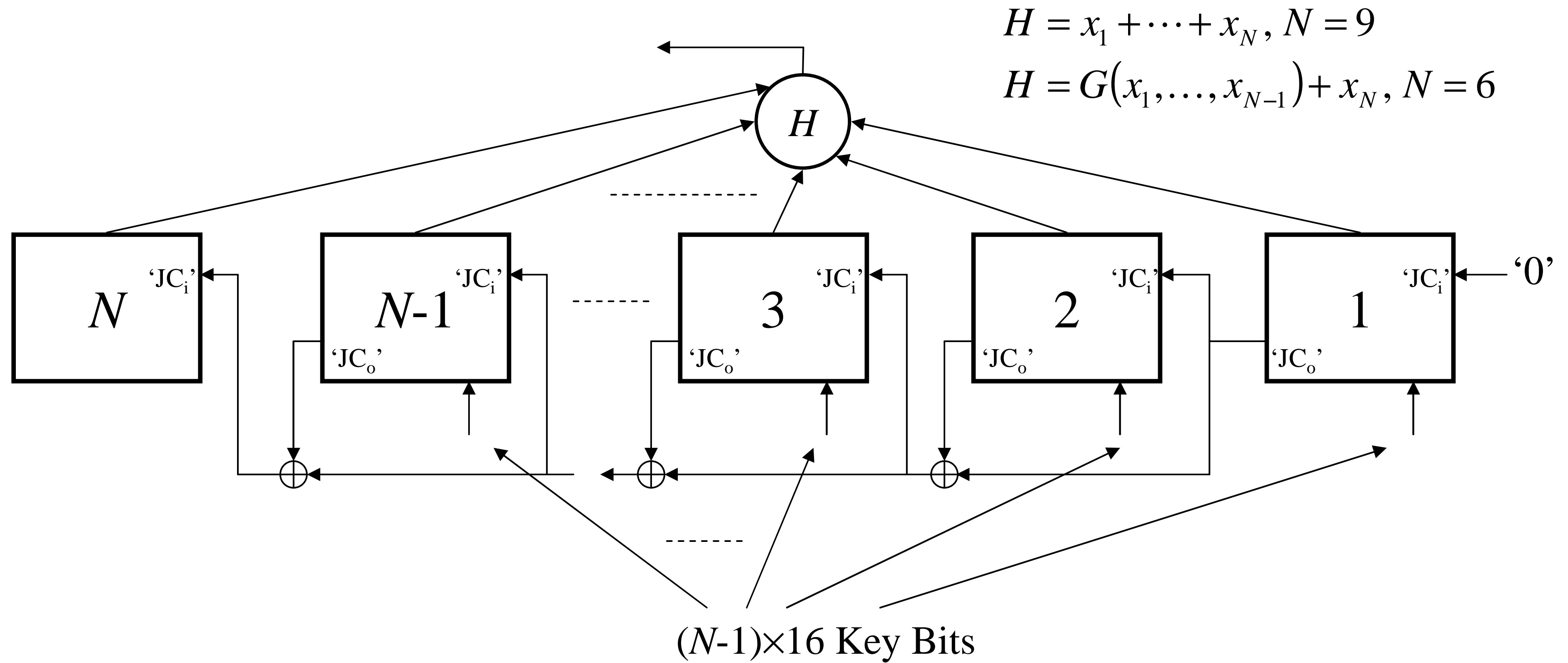
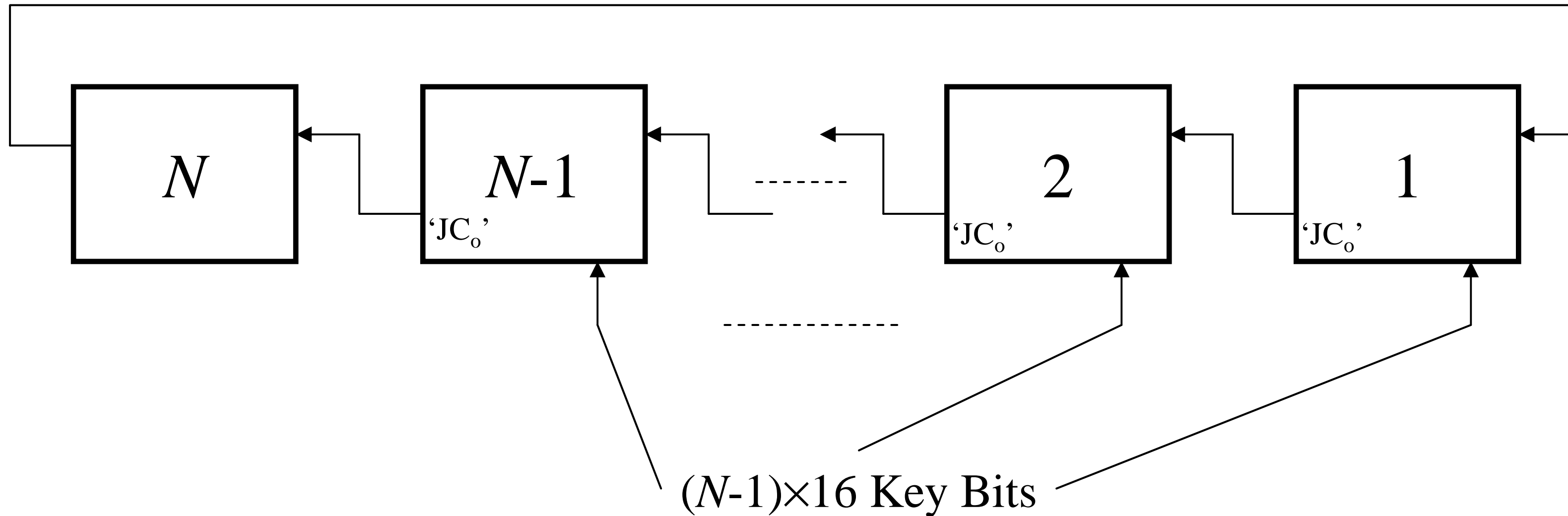


Fig. 4 Key-Stream Generation Mode



'Accumulated' Cascade Jump Control

Fig. 5 Shift Mode (Initialization)



- Load $N\times 18$ register cells with binary expansion of π
- Make 128 steps in Shift Mode
- Save contents of $N\times 18$ register cells (Initialization Vector) for later use in IV mode