LOCK-IN EFFECT IN CASCADES OF CLOCK-CONTROLLED SHIFT-REGISTERS

William G Chambers[1)] Dieter Gollmann[2)]

[1)]Department of Electronic and Electrical Engineering,
King's College (KQC), Strand,
London WC2R 2LS, United Kingdom

[2)]Fakultät für Informatik, Universität Karlsruhe,
Technologie-Fabrik Karlsruhe, Haid-und-Neu-Strasse 7,
7500 Karlsruhe 1, W Germany.

ABSTRACT

Cascaded cryptographic keystream generators as proposed by Gollmann possess a cryptanalytic weakness termed "lock-in" in this article. If the initial state has been guessed correctly apart from its phase a decryption cascade can be set up in which the effects of each stage of the original cascade are unravelled in reverse order. Once the decryption cascade has "locked in" on the original cascade, the state of the latter is known, and hence its future output and its output in the remote past. This weakness is studied; its effects are readily mitigated by taking certain precautions. Lock-in may also be used constructively as a synchronization technique.

I. INTRODUCTION

Cryptographic binary sequences produced with the aid of shift-registers have been much studied in the open literature over the last twenty years. An important parameter is the linear equivalence, which measures the resistance of a sequence generator to attacks using linear algebra [1, p199]. A good discussion of ways of increasing the linear equivalence is given by Rueppel [9].

One method is to use a non-linear function to combine the simultaneous outputs of several shift-registers. The use of clock-controlled shift-registers has also been proposed by several authors [2, 5, 12, 13]. Typical of such systems is a cascade of clock-controlled shift-registers [6]. The periods and linear equivalences are readily made very large, and the statistical properties of at least the original versions have been proved to be good [7].

The fact that these systems are readily designed to have a high linear equivalence and hence be immune against the algebraic attack does not preclude other types of weakness. Thus attacks on sequences produced by non-linear combining functions have been studied by Siegenthaler [10, 11]. In this article a weakness which may occur in systems using clock-controlled shift registers is examined. This weakness can readily be guarded against by taking suitable precautions; nonetheless the user should be made aware of the possibility, since the weakness is not obvious. Of course this does not guarantee that there are no other hazards.

The cryptanalytic problem is the following: Assume that an enemy knows a) the construction of the generator and b) a large number of consecutive bits of the output, which for the sake of definiteness will be assumed to start at the beginning of the sequence. Then with limited computing resources can he deduce the initial setting of the generator, or at least the future output?

II. THE CASCADE GENERATOR

The keystream generator proposed in [6] consists of a number of stages, K say, each like that shown in Fig 1. The main component of each stage is a clock-controlled cycling register (CR) of length p, this length being the same for each stage. If regularly clocked (or stepped), CR produces an endless repetition $\mathbf{b}^{\infty}$ of the binary sequence $\mathbf{b}=\{b(0), b(1), \cdots b(p-1)\}$, where the $b(i)$ are determined by the initial setting for this stage. (The only restriction on $\mathbf{b}$ is that $\mathbf{b}^{\infty}$ should have shortest period p. Thus with $p=3$ the choices $\mathbf{b} = \{000\}$ and $\mathbf{b} = \{111\}$ are excluded for then $\mathbf{b}^{\infty}$ has period 1.) The binary input a_t is added (*mod* 2) to the output of CR to give the output c_t of this stage, which then becomes the input of the next. The binary input also causes CR to be stepped (afterwards) if $a_t=1$, but not if $a_t=0$. The "slight delay" is put in the figure to emphasise that the step takes place after addition, that is, the rule is "add then step". We shall say that the stage *uses* the sequence $\mathbf{b}$. The input to the first stage is 111... . The output of the final stage is the output of the generator.

The sequences $\{c_t\}$ and $\{a_t\}$ of the stage in Fig 1 are related by

$$c_t=a_t+b(S_{t-1}) \ mod\ 2,\ S_t=S_{t-1}+a_t \ mod\ p,\ t=0,1,2,... \tag{1a}$$

with the initial condition

$$S_{-1}=0. \tag{1b}$$

Evidently S_t is the sum $\sum_{t'=0}^{t} a_{t'} \ mod\ p$. Since it determines where CR has got to in its cycle it will be called the *phase* of CR. (By $a \ mod\ p$ for positive p we mean the value x satisfying $0 \le x < p$ obtained by adding (subtracting) a suitable integer multiple of p to (from) a.)

A modified system (the "m-sequence cascade") consists of a similar cascade of clock-controlled linear feedback shift registers of length n with primitive feedback polynomials [1, p187]. The regularly clocked output of such a register has period $p=2^n-1$, and the sequence $\{b(0), b(1), \cdots b(p-1)\}$ is a period of the m-sequence.

The output of a Gollmann cascade of length K has period p^K if p is an odd prime [6]. If p satisfies a further fairly weak condition (that (2^j-1) is not a multiple of p for any j satisfying $0<j<p-1$) then the linear equivalence is either p^K or p^K-1 [6, 4]. Among the small primes 3, 5, 11, 13, 19 and 29 satisfy this condition whereas 7, 17, 23, and 31 do not. In an m-sequence cascade of length K the period is $(2^n-1)^K$ and the linear equivalence exceeds $n(2^n-1)^{K-1}$ [3].

III. THE ATTACK

We now suppose that the stage just described is the final stage of the generator, so that $\{c_t\}$ is the final output, some of which has been intercepted by the cryptanalyst X. (How much he needs is considered below.) In the attack to be described he tries to reverse the transformation from $\{a_t\}$ to $\{c_t\}$ effected by the final stage. Iteration of this technique should then enable him to "unravel" the cascade, starting with the final stage.

The reversing transform is carried out as follows: X guesses a sequence b′ and a value S'_{-1}, and then sets

$$a'_t=c_t-b'(S'_{t-1}) \ mod\ 2,\ S'_t=S'_{t-1}+a'_t \ mod\ p,\ t=0,1,2,... \tag{2}$$

where the primed quantities are guesses or deductions from guesses. (When

$b'=b$ and $S'_{-1}=S_{-1}$ we find that $\{a'_t\}=\{a_t\}$.) Such a transform may be implemented by a decryption stage (Fig 2) using the sequence $\mathbf{b}'$ with initial phase S'_{-1}. In the case when $b'(t)=b((t+\phi) \bmod p)$ for some ϕ we say that b has been guessed correctly except for phase. (Thus for $p=3$ there are only two non-trivial choices for $\mathbf{b}$ differing by more than phase.)

We now make Assumption A (to be examined below): Suppose that X has guessed the sequence $\mathbf{b}'$ correctly except possibly for the phase. Let c_t in (2) be the output from (1). We may instead presume that $\mathbf{b}'=\mathbf{b}$ and that the initial guess S'_{-1} needed for (2) may be incorrect. Then as the iteration (2) proceeds the phase S'_t may be expected to bounce around in some manner until it happens to take the correct value S_t. Thereafter it will be *locked in* into its correct value, so that for all future t we find $S'_t=S_t$ and $a'_t=a_t$. (Investigations described in Sec 4 indicate that this takes a number of steps roughly equal to $\frac{1}{2}p^2$ on average.)

When the whole cascade is unravelled, the original input 111... is recreated. This is how X knows whether he has succeeded. At the same time he learns the phase of each CR in the generating cascade, not, it is true, at the start $t=0$, but at a value of t (t_0 say) where it is fairly safe to assume that lock-in has taken place. Thus the output from the generator after t_0 can be predicted. It is also possible to work backwards from t_0 to $t=0$, so that the initial setting can be deduced. Let us consider (1a) as applying to the first stage of the generator, where X knows the input a_t for all t (as 1). Let us suppose moreover that X knows S_{t-1} for $t>t_0$. Then he may find S_{t-2} as $S_{t-1}-a_{t-1} \bmod p$, and so proceed backwards to S_{-1}. Thus the c_t may also be found all the way back to the start. But $\{c_t\}$ is the input to the second stage, and thus the process can be iterated.

Assumption A is now examined. There are situations where it is valid for every stage without further ado: a) If for ease of manufacture the contents of each CR are laid down in advance, with the key determining how many steps are taken by each CR in preparing the initialization, then X knows each CR except for phase. b) In the m-sequence cascade with registers of length n the period of each register is $p=2^n-1$. If the feedback polynomial of each stage is specified in manufacture, the outputs are again known apart from their phase, since all m-sequences associated with a given primitive feedback polynomial are cyclic shifts of one another [1, p186].

In other cases X has to make a number of trials, in only one of which Assumption A is valid for every stage. Thus in Gollmann's cascade with p prime there are 2^p-2 initial settings for CR, and $(2^p-2)/p$ initial settings that differ by more than phase. For a cascade of length K the number of

possible trials is thus $((2^p-2)/p)^K$, that is 2^K with $p=3$.

IV. NUMBER OF STEPS NEEDED

In this section the number of steps needed to achieve lock-in is discussed, firstly just for the final stage, and then for the whole cascade. Assumption A is taken as valid for every stage. Evidently this number is also the minimum length of the sequence needed for the attack described in Section 3.

The number of steps needed on average to get a decryption stage (using the correct sequence apart from phase) to lock-in to the final stage of a cascade can be estimated as follows. The previous stages of the cascade are regarded as a random binary generator G. The output $\{a_t\}$ of G is then passed through the final encryption stage E to produce an output $\{c_t\}$ according to (1). The sequence $\{c_t\}$ is then passed through the decryption stage D to produce an output $\{a'_t\}$ according to (2). The stage D uses the same sequence **b** as is used by E, but the initial phases may not agree. Until lock-in is achieved the input to D will be regarded as random, and so the difference of the phases $\Delta_t = S_t - S'_t$ behaves as though in the problem of the random walk [8, p213], either increasing or decreasing by unity with equal probability, or staying the same. Initially Δ_t is taken to have any value between 0 and $p-1$ with equal probability, so that its mean is approximately $\frac{1}{2}p$. Lock-in takes place when Δ_t reaches either of the values 0 or p. For a random walk to cover a distance d requires a number of steps of the order of d^2, and so in this case we may expect the mean number of steps needed to achieve lock-in to be of order p^2.

This conclusion is borne out for p up to 31 by the more careful treatment described in the appendix. The mean μ_p and standard deviation σ_p of the number of steps to lock-in for a single stage have been computed for p taking the prime values from 3 to 31 to give the results shown in Table 1, which lists the values $\mu'_p = \mu_p / p^2$ and $\sigma'_p = \sigma_p / p^2$. The results are approximate to about 6 percent for $p \geq 19$.

TABLE 1

p	μ'_p	σ'_p		p	μ'_p	σ'_p
3	0.3210	0.4669		17	0.5365	0.6799
5	0.4997	0.6914		19	0.52	0.65
7	0.5681	0.7957		23	0.53	0.66
11	0.5645	0.7643		29	0.52	0.63
13	0.5538	0.7317		31	0.51	0.62

Complete lock-in for the whole cascade $E_1, E_2, \cdots E_K$ (with K the number of stages) requires a similar cascade of decryption stages $D_1, D_2, \cdots D_K$, with D_k having the same sequence as E_k. The output from D_k is the input to D_{k-1}. By an iterative argument starting with $k = K$ it is evident that once D_k has locked in on E_k the input to D_{k-1} is the same as the output from E_{k-1}, and so D_{k-1} can start to lock-in on E_{k-1}. It is conceivable that D_{k-1} might already have started to lock in on E_{k-1} before D_k had locked in properly on E_k, but we shall assume that each lock-in starts with random initial conditions as soon as the previous stage has locked in. Thus the number of steps needed to achieve over-all lock-in is the sum of K independent identically distributed random variables, and so its mean is $K\,\mu'_p\,p^2$, and its standard deviation is $K^{1/2}\sigma'_p\,p^2$.

Computer simulations (for p = 3, 5, 11, and 13) bear out these conclusions. The only surprise was that for p = 5, 11 and 13 in about 10% of the cases D_1 and D_2 failed to lock-in. This is presumably because the input 111... to E_1 can hardly be regarded as random. Although this may be an embarrassment to the cryptanalyst it is probably not a serious obstacle.

V. USE OF 'STEP THEN ADD'

It might appear that the arrangement where the "slight delay" of Fig 1 is put instead at the point X would give a different problem, with a_t implicitly dependent on c_t, rather than explicitly as in (2). For then we have

$$c_t = a_t + b(S_t) \bmod 2,\ S_t = S_{t-1} + a_t \bmod p. \tag{3}$$

Appearances are however deceptive, and the inversion may be carried out by

$$a_t = c_t - b(S_t) \bmod 2,\ S_{t-1} = S_t - a_t \bmod p, \tag{4}$$

where we let t run *downwards* from some large value N to 0, and all we need to guess is the initial value S_N. Thus lock-in can be made to occur if the output sequence from (3) is fed backwards into (4).

This suggests that if the cryptographer arranges that a choice between "add then step" and "step then add" be made for each stage under the control of the key, then the use of lock-in as a cryptanalytic technique is made more difficult. It may however be better to spend the additional cryptographic effort on extending the length of the cascade, with a corresponding increase in the linear equivalence and the period [6].

VI. GUARDING AGAINST CRYPTANALYSIS BY LOCK-IN

First suppose the validity of Assumption A. Then the length of the bit-string needed for the attack by lock-in is of the order of $S = Kp^2$, where p is the length of the cycling sequence **b** and K is the number of stages in the cascade. Since the decryption involves passing the string through K decryption stages the computing complexity, that is the number of computing steps needed, is of the order of $C_A = K^2p^2$. If on the other hand Assumption A is not valid then every possible instance of **b** has to be tried in each stage and so the computing complexity is of the order of $C = K^2p^2.((2^p - 2)/p)^K$. To give examples of these values we note that C exceeds 10^{21} for $p = 3$, $K = 56$, or for $p = 11$, $K = 8$, with S less than 1000 in both cases.

For an m-sequence cascade we set $p = 2^n - 1$ where n is the register length. It may be necessary to use fixed feedback connections, so that Assumption A is valid. Then we find that $C_A > 10^{21}$ for $n = 34$, $K = 2$, or for $n = 29$, $K = 59$. Huge string-lengths are needed in these cases. We find $S = 5.9\times10^{20}$ and 1.7×10^{19} respectively. On the other hand small values of n would not be safe.

Without Assumption A the attack may be improved by a "meet-in-the-middle" technique. The encryption cascade is regarded as being in two sections, of length a at the top and b at the bottom, with $a + b = K$. All $(2^p - 2)^a$ possible initializations of the top section are tried and the initial part of each sequence thus generated is stored in order, together with the setting that generated it. All $((2^p - 2)/p)^b$ initializations of the lower part are used in a decryption cascade of length b to lock-in on to the sequence to be broken. Again the output strings are ordered. Then the analyst looks for matching pairs in the two ordered lists. If a matching pair is found it is

investigated further. Optimally the two lists should be roughly of the same size, so that for small values of p the size of b is around two-thirds to three-quarters of K. This value should perhaps replace K in the above considerations.

VII. USE OF LOCK-IN FOR SYNCHRONIZATION

So far it has been assumed that the cascade is used as a pseudo-random binary sequence generator, with the all-1's sequence fed in at the top. Under these conditions lock-in is a cryptanalytic hazard. However it may be employed more constructively by the cryptographer. Suppose that the plaintext is fed into the top of the cascade, and the ciphertext taken from the bottom. Then the legitimate receiver will use a decryption cascade. Here the key given to the receiver specifies the contents of each register and Assumption A is certainly satisfied. Then it is almost certain that the lock-in property ensures the self-synchronization of the decryption, even if it is not properly synchronized at any stage. Under these circumstances we would want fairly quick lock-in, so that short registers (say $p=3$) would be used in a long cascade (say $K=100$). A long cascade is of course vital for security, the effective keylength being K bits with $p=3$. The mean time to lock-in with $p=3$ and $K=100$ is about $0.3210\times 3^2\times 100=290$ steps.

We have also studied the effects of a single-bit error on lock-in. There are three types of such an error, the alteration, the insertion and the loss of a bit. Computer simulations (carried out for $p=3, 5, 7$ and 11 with $K=31$) suggest that lock-in times after a single-bit error have a distribution very like that for lock-in starting with random phases. Thus for the cascade with $p=3$ and $K=100$ the mean recovery time would be around 290 steps. This is just over twice the recovery time for a 64-bit block cipher such as DES [1, p267] used in the cipher-feedback mode [1, p287]. Moreover as far as a cascade cipher is concerned the loss or insertion of a bit is no worse than the alteration of a bit, whereas for a block cipher such an error causes misalignment of the blocks, and some method for maintaining synchronization is needed.

APPENDIX: Number of steps for lock-in of a single stage

We develop further the model of Sec 4 in which a random binary input $\{a_t\}$ is fed into an encryption stage E using a given sequence **b** of given least period p, and the output $\{c_t\}$ generated according to (1) is fed to a decryption stage D also using **b**. We find easily computed expressions for the mean and variance of the number of steps to lock-in for any given **b**, averaged over the initial states of D and E. By a random binary sequence $\{a_t\}$ we mean that the a_t are independent identically distributed random variables taking just the values 0 and 1 with equal probabilities, or equivalently that for any n all sequences of length n are equally likely. Since the sequences $\{a_t\}$ and $\{c_t\}$ (for given **b** and S_{-1}) are in one-to-one reciprocal correspondence it is readily shown that $\{c_t\}$ is also a random binary sequence in the above sense.

Equations (1a) and (2) may be written as

$$S_t = ((c_t + b(S_{t-1}))\ mod\ 2) + S_{t-1}\ mod\ p, \tag{5a}$$

$$S'_t = ((c_t + b(S'_{t-1}))\ mod\ 2) + S'_{t-1}\ mod\ p. \tag{5b}$$

Lock-in occurs as soon as $S_t = S'_t\ mod\ p$. The value pair (S_t, S'_t) specifies the state of the system at time t. We first show that, starting from any state, lock-in can take place with non-zero probability after $p(p-1)$ steps. This result will be used to show that lock-in takes place eventually with probability one, and it guarantees the convergence of the theory below, as well as the existence of the mean and variance of the time to lock-in. To do this we suppose that $\{a_t\}$ happens to be the all-ones sequence. Then by (1a) S_t increases by 1 on every step (*mod p* of course). Now suppose that lock-in does not take place. Then beyond some step t_0 the quantity S'_t must keep some fixed distance s ahead $(0 < s < p)$, so that $S'_t = S_t + s\ mod\ p$ for $t > t_0$. Then from (5) it follows that $b((i+s)\ mod\ p) = b(i)$ for all i such that $0 \le i < p$, and so $\mathbf{b}^\infty$ has a period less than p, in fact the highest common factor of s and p. This contradicts the assumption that p is the least period. This catching up needs at most $p(p-1)$ steps. For S_t must gain on S'_t by at least 1 every time it goes round the cycle $(0, 1, , \cdots p-1)$. However S'_t cannot be more than $p-1$ ahead of S_t at the beginning, and hence the result.

To compute the mean and variance we use a state-transition matrix T whose rows and columns are labelled by states of the system. The "coalesced" states (with $S_t = S'_t$) need not be included among these, and there

is no need to distinguish between S_t and S'_t, so the states may be represented as number pairs (a,b) with $0 \le a < b < p$, the numbers being of course values of S_t and S'_t. There are altogether $\frac{1}{2}p(p-1)$ such states, and they will be denoted by Greek suffices α, β and γ. Let $T_{\beta\alpha}$ denote the probability of a transition from α to β. Then we find that $T_{\beta\alpha} \ge 0$, and that $\sum_\beta T_{\beta\alpha} \le 1$ with $\sum_\beta T_{\beta\alpha} < 1$ if α can go to a coalesced state in one step. Let $p_\alpha(t)$ denote the probability of the system being in the state α at step t. We find $p_\beta(t+1) = \sum_\alpha T_{\beta\alpha} p_\alpha(t)$ or in vector-matrix notation $\mathbf{p}(t+1) = \mathbf{T}\mathbf{p}(t)$, so that $\mathbf{p}(n) = \mathbf{T}^n \mathbf{p}(0)$. The probability of "no lock-in after n steps" may be written as $P_n = \mathbf{e}'\mathbf{p}(n)$ where $\mathbf{e}$ is the all-ones vector. With a start from any state α, lock-in takes place with a probability not less than $\lambda = 2^{-Q}$ after $Q = p(p-1)$ steps. (The quantity λ is the probability that $\{a_t\}$ starts with Q consecutive 1's.) Now the probability distribution after n steps starting from the state α is $p_\beta = (\mathbf{T}^n)_{\beta\alpha}$, so that $\sum_\beta (\mathbf{T}^Q)_{\beta\alpha} \le 1 - \lambda$. Thus for any integer $l \ge 0$ we find

$$\sum_\beta (\mathbf{T}^{(l+1)Q})_{\beta\alpha} = \sum_\gamma \left(\sum_\beta (\mathbf{T}^Q)_{\beta\gamma}\right) (\mathbf{T}^{lQ})_{\gamma\alpha} \le (1-\lambda) \sum_\gamma (\mathbf{T}^{lQ})_{\gamma\alpha}.$$

By iteration this is then less than or equal to $(1-\lambda)^{l+1}$, and hence so is each term in the sum on the left. We are using the fact that all these matrix components are non-negative. Thus we find that $\mathbf{T}^n \to \mathbf{O}$ as $n \to \infty$. From this it follows (by *reductio ad absurdum*) that the eigenvalues of $\mathbf{T}$ are strictly less than unity in magnitude. This approach may well give a hopelessly pessimistic estimate of the rate of convergence of $\mathbf{T}^n$ to $\mathbf{O}$, but it is all that is needed for the theory.

The initial probability distribution will be taken as uniform, with $\mathbf{p}(0) = (2/p^2)\mathbf{e}$; this takes account of the possibility of coalescence at the start, since $P_0 = \mathbf{e}'\mathbf{p}(0) = 1 - 1/p$. The mean time to coalescence is then given by

$$\mu = \sum_{n=0}^{\infty} (n+1)(P_n - P_{n+1})$$

since a fraction $P_n - P_{n+1}$ coalesces at step $n+1$. Thus we find that

$$\mu = \sum_{n=0}^{\infty} P_n = (2/p^2) \sum_{n=0}^{\infty} \mathbf{e}'\mathbf{T}^n\mathbf{e} = (2/p^2)\mathbf{e}'(\mathbf{I}-\mathbf{T})^{-1}\mathbf{e}$$

where $\mathbf{I}$ is the unit matrix. Here a matrix geometric progression has been summed, which is possible since all the eigenvalues are less than one in magnitude.

In like manner the mean square time to coalescence is given by

$$\nu = \sum_{n=0}^{\infty} (n+1)^2 (P_n - P_{n+1})$$

from which we find

$$\nu = (2/p^2)\mathbf{e}'(\mathbf{I}+\mathbf{T})(\mathbf{I}-\mathbf{T})^{-2}\mathbf{e}.$$

For reasonable values of p (say up to 31) these computations are not too hard. They involve the solution of linear equations rather than matrix inversions, and they are assisted by the facts that T is sparse, with all the non-zero elements equal to ½, and that it is a banded matrix if the states are ordered by increasing separation of the locations.

As an example we consider a case with $p=5$. The matrix T is then of size 10×10. The states used for labelling are preferentially ordered as 01, 12, 23, 34, 04, 02, 13, 24, 03, 14. Here 01 stands for (0, 1) etc. With $\mathbf{b}=\{0,1,1,0,1\}$ the possible transitions $\alpha\to\beta$ are 01→02, 12→12, 12→23, 23→24, 34→03, 04→14, 02→12, 02→03, 13→23, 13→14, 24→03, 24→24, 03→03, 03→14, 14→14, 14→02. For these $T_{\beta\alpha}$ is ½. The other elements of T are zero.

The final part of the calculation is to average μ and ν over all possible **b** with p specified. These averages are denoted by μ_p and ν_p. The standard deviation σ_p of the lock-in time is given by $\sigma_p^2 = \nu_p - \mu_p^2$. Since for $p \geq 19$ the number of instances of **b** is rather large (being equal to $(2^p - 2)/p$), the computations were restricted to averaging over 300 quasi-random choices, giving an accuracy of a few percent.

REFERENCES

[1] H Beker, F Piper, *Cipher Systems: The Protection of Communications,* (New York: Wiley) 1982

[2] T Beth, F C Piper, "The Stop-and-Go Generator", *Advances in Cryptology: Proceedings of Eurocrypt 84* (T Beth, N Cot, I Ingemarsson, eds) Lecture Notes in Computer Science 209, 88-92 (Berlin: Springer-Verlag) 1985

[3] W G Chambers "Clock-controlled Shift Registers in Binary Sequence Generators", IEE Proc E, 1988, 135, 17-24

[4] W G Chambers and D Gollmann, "Generators for Sequences with Near-maximal Linear Equivalence", IEE Proc E, 1988, 135, 67-69

[5] W G Chambers, S M Jennings, "Linear Equivalence of Certain BRM Shift Register Sequences", Electronics Letters, 1984, 20, 1018-1019

[6] D Gollmann, "Linear Recursions of Cascaded Sequences" *Contributions to General Algebra 3, Proceedings of the Vienna Conference June 1984* (Verlag Holder-Pichler-Tempsky, Wien 1985 - Verlag B G Teubner, Stuttgart)

[7] D Gollmann, "Pseudo Random Properties of Cascade Connections of Clock Controlled Shift Registers" in *Advances in Cryptology, Proceedings of Eurocrypt 84,* (ed T Beth, N Cot, I Ingemarsson) Lecture Notes in Computer Science 209, pp93-98 (Berlin: Springer Verlag 1985)

[8] A Papoulis, *Probability, Random Variables, and Stochastic Processes* 2nd ed, (Singapore: McGraw-Hill) 1984

[9] R A Rueppel, *Analysis and Design of Stream Ciphers,* (Heidelberg: Springer-Verlag) 1986

[10] T Siegenthaler, "Correlation Immunity of Nonlinear Combining Functions for Cryptographic Applications", IEEE Trans Info Theory, 1984, IT-30, 776-780

[11] T Siegenthaler, "Decrypting a Class of Stream Ciphers Using Ciphertext only", IEEE Trans Computers, 1985, C-34, 81-85

[12] B Smeets, "A Note on Sequences Generated by Clock Controlled Shift Registers", *Advances in Cryptology: Eurocrypt '85,* (F Pichler ed), Lecture Notes in Computer Science 219, pp142-148 (Berlin: Springer-Verlag) 1986

[13] R Vogel, "On the linear complexity of cascaded sequences", *Advances in Cryptology: Proceedings of Eurocrypt 84* (T Beth, N Cot, I Ingemarsson, eds) Lecture Notes in Computer Science 209, 99-109 (Berlin: Springer-Verlag 1985)

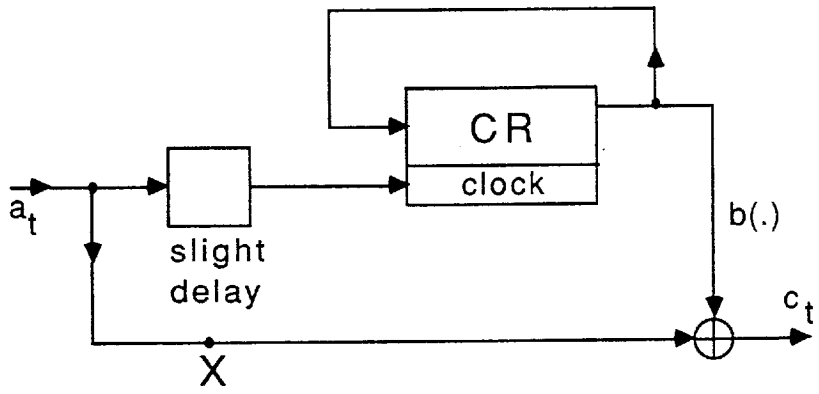

FIG 1: A stage of Gollmann's cascade, as described in Sec 2. The input bit a_t is added to the output from the cycling register *CR* to give the output c_t. It is also used to clock *CR* *after* the addition. In another arrangement (Sec 5) the "slight delay" is put at *X* instead, so that *CR* is clocked *before* the addition.

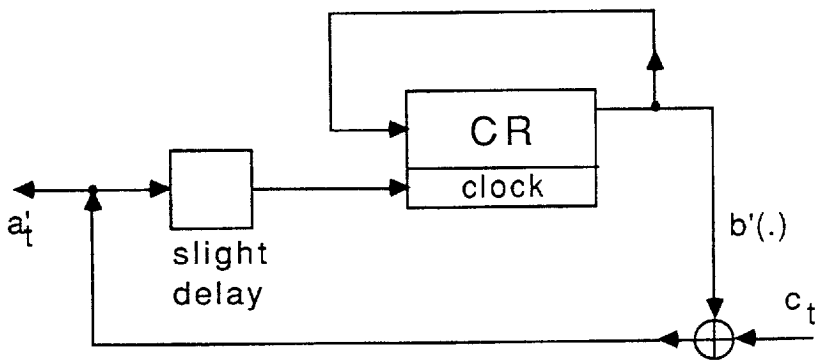

FIG 2: A decryption stage for reversing the transformation accomplished by the stage in Fig 1. Here the "slight delay" prevents a race round the loop.