Cryptanalysis of Sfinks[⋆]

Nicolas T. Courtois

Axalto Smart Cards Crypto Research, 36-38 rue de la Princesse,
BP 45, F-78430 Louveciennes Cedex, France, `courtois@minrank.org`

Abstract. Sfinks is an LFSR-based stream cipher submitted to ECRYPT call for stream ciphers by Braeken, Lano, Preneel *et al.* The designers of Sfinks do not to include any protection against algebraic attacks. They rely on the so called “Algebraic Immunity”, that relates to the complexity of a simple algebraic attack, and ignores other algebraic attacks. As a result, Sfinks is insecure.

Key Words: algebraic cryptanalysis, stream ciphers, nonlinear filters, Boolean functions, solving systems of multivariate equations, fast algebraic attacks on stream ciphers.

1 Introduction

Sfinks is a new stream cipher that has been submitted in April 2005 to ECRYPT call for stream cipher proposals, by Braeken, Lano, Mentens, Preneel and Varbauwhede [6]. It is a hardware-oriented stream cipher with associated authentication method (Profile 2A in ECRYPT project).

Sfinks is a very simple and elegant stream cipher, built following a very classical formula: a single maximum-period LFSR filtered by a Boolean function. Several large families of ciphers of this type (and even much more complex ones) have been in the recent years, quite badly broken by algebraic attacks, see for exemple [12, 13, 1, 14, 15, 2, 18]. Nevertheless the specialists of these ciphers counter-attacked by defining and applying the concept of Algebraic Immunity [7] to claim that some designs are “secure”. Unfortunately, as we will see later, the notion of Algebraic Immunity protects against only one simple algebraic attack and ignores other algebraic attacks. More realistic (but also more complex to apply) security criteria for stream ciphers have been proposed in [13, 16].

We note that it is possible to design stream ciphers that would be in some sense “protected” against algebraic attacks (and also in a similar way against other known attacks, such as correlation and fast correlation attacks). It is even a common practice to add to the stream cipher some components that would make all these attacks less efficient (cf. [26, 13]), or even clearly impractical (e.g. [8]) to apply. This can be done, for example, with irregular clocking and/or a final compression component that would combine several consecutive outputs of the Boolean function in a complex way. The drawback of this is (possibly) some loss in speed and an important loss in hardware footprint (for example, irregular clocking would require a large buffer). It would make the design less elegant, and also much more vulnerable to timing and side-channel attacks.

Finally, the designers of Sfinks choose to stick to this simple and elegant design that is easy to study (and for which we would like to thank the authors as it helps the cryptanalysts too !). We have a simple filtered LFSR on which most of the known attacks can be applied directly and the only thing that prevents these attacks so far, is the parameters of Sfinks [6] were chosen so that the attack complexity is close to 2^{80}, without even any margin for (frequent) algorithmic improvements. Consequently, it is possible to say that Sfinks has been designed with no “protection” whatsoever against known attacks.

[⋆] This work was partially supported by the French Ministry of Research RNRT X-CRYPT project and by the European Commission via ECRYPT network of excellence IST-2002-507932.

2 Short Description of Sfinks

A regularly clocked 256-bit binary LFSR provides, at each clock, 17 out of 256 its state bits, that are supplied to a Boolean function. The keystream is composed of successive output bits of this Boolean function.
The LFSR used in Sfinks is described by the following recursion formula:

$$\begin{aligned} s_{t+256} = {} & s_{t+212} \oplus s_{t+194} \oplus s_{t+192} \oplus s_{t+187} \oplus s_{t+163} \oplus s_{t+151} \oplus s_{t+125} \oplus \\ & s_{t+115} \oplus s_{t+107} \oplus s_{t+85} \oplus s_{t+66} \oplus s_{t+64} \oplus s_{t+52} \oplus s_{t+48} \oplus s_{t+14} \oplus s_t \end{aligned}$$

The Boolean Function Used in Sfinks
We call $f(x_t^{16}, \ldots x_t^0)$ the output filtering function of Sfinks [6]. The 17 variables used are selected as follows among the state bits of the LFSR:

$$\begin{aligned} \left(x_t^{16}, \ldots, x_t^0\right) \overset{def}{=} \big(& s_{t+255}, s_{t+244}, s_{t+227}, s_{t+193}, s_{t+161}, s_{t+134}, s_{t+105}, s_{t+98}, s_{t+74}, \\ & s_{t+58}, s_{t+44}, s_{t+21}, s_{t+19}, s_{t+9}, s_{t+6}, s_{t+1}, s_t \big) . \end{aligned}$$

We define as P the corresponding projection mapping $GF(2^{256}) \rightarrow GF(2^{17})$. P is a multivariate linear transformation.
The function f is a highly non-linear Boolean function of degree 15, with 17 variables. It is defined as follows:

$$z_t = f(\bar{x}_t) = f(x_t^{16}, \ldots, x_t^0) = (INV(x_t^{16}, \ldots, x_t^1)\&1) \oplus x_t^0.$$

with INV being the inverse in $GF(2^{16})$ defined as follows. Let $GF(2^{16})$ be defined as $GF(2)[Z]/Z^{16} + Z^5 + Z^3 + Z^2 + 1$. We define INV as inverse in $GF(2^{16})$ complemented by $0 \mapsto 0$ as in Rijndael, implemented as a table operating on 16-bit words, in such a way that the least significative bit of the input is x_t^1, and it corresponds to the coefficient of Z^1 in polynomial arithmetic modulo $Z^{16} + Z^5 + Z^3 + Z^2 + 1$. At the output, f is defined by the least significative bit of the output word of INV (which again corresponds to Z^1).

3 Algebraic Attacks on Sfinks

Algebraic attacks on stream ciphers are based on the following observation. Let $L : GF(2^n) \rightarrow: GF(2^n)$ be a multivariate linear transformation that corresponds to clocking the LFSR. For each i, L^i is a known multivariate linear transformation. At any moment t in the cipher history, all bits of the internal state are **known** linear combinations of the bits of the initial state $s_0, \ldots, s_n$ (for Sfinks $n = 256$).
Let $z_t, t = 0, 1, 2, \ldots$ be the keystream generated by Sfinks and let f be its output Boolean function. We recall that $f \circ P$ is the version of f defined from $GF(2^n) \rightarrow GF(2)$ and takes all n bits as inputs, 17 of which are used and the other are ignored. Then we can write the problem of key recovery in Sfinks as follows.

$$\begin{cases} z_0 = f(\ P \quad (s_0, \ldots, s_{n-1}) \ ) \\ z_1 = f(\ P\,(L \ \ (s_0, \ldots, s_{n-1}))\ ) \\ \qquad \vdots \\ z_t = f(\ P\,(L^t\ (s_0, \ldots, s_{n-1}))\ ) \\ \qquad \vdots \end{cases} \qquad (\#)$$

Algebraic attacks on stream cipher work by solving, (by more or less sophisticated methods) the above system equations (or a part of it). They use extensively the fact that the degree of these and other derived algebraic equations is preserved by the linear operation $P \circ L^t$, at any moment t.
In this paper we use the terminology of [13] to classify algebraic attacks on stream ciphers as S1, S2, S3, S4, S5 and S6. In addition we will give a complete description of all proposed attacks on Sfinks.

3.1 First Basic Algebraic Attacks on Sfinks (S1 and S2)

The simplest attack scenario we can think of is known as direct linearization attack or S1, see [12, 13, 3]. It works as follows: the equations are of degree 15, and if we dispose of about $T = \binom{n}{15} + \binom{n}{14} + \ldots + \binom{n}{0} \approx 2^{79.2}$ keystream bits, than we can rewrite the system (#) as a system of T linear equations with T variables - all monomials are treated as new variables. The system is the solved with the complexity of T^{ω}, with ω being the exponent of the Gaussian reduction. In theory it is at most $\omega \leq 2.376$, see [9]. However the (neglected) constant factor in this algorithm is expected to be very big. Thus, in this paper we will systematically estimate the complexity of solving linear systems as about $T^{log_2 7}$ operations, which is believed to be achievable in practice with the Strassen's algorithm [29].
With S1 attack we get a very large complexity: $T^{log_2 7} \approx 2^{222}$.
Probabilistic Variant. In scenario S2, introduced in [12], the Boolean function is approximated by a function of a lower degree to get a lower attack complexity. We do not develop tools to find good low-degree approximations of f. Nevertheless we believe that it is very unlikely that f used in Sfinks has very good approximations that would lead to efficient algebraic attacks. The approximation to be interesting must hold with probability very close to 1, and in [12] such approximations existed because there were very few monomials of high degree. In Sfinks, f has many monomials of very high degree.

3.2 The S3 Algebraic Attack on Sfinks

In the scenario S3 introduced by Courtois and Meier in [13] and also studied by Carlet *et al* in [7], the degree of the equations (#) is substantially reduced.
This is possible due to the existence of a low-degree algebraic relation that relates input and output bits of f. For example, we assume that there exists an equation of the type:

$$zX^d + X^d.$$

This notation is very compact and convenient and is taken from [10]. It means that there is (at least one) equation of type

$$z \cdot g(x^{16}, \ldots, x^0) + h(x^{16}, \ldots, x^0) = 0 \qquad \text{with } z = f(x_t^{16}, \ldots, x_t^0).$$

with g and h being some multivariate polynomials of degree up to d. The equation has to be true with probability 1, i.e. for every choice of $(x^{16}, \ldots, x^0)$. In [7] it is shown[1] that

[1] In fact this equivalent formulation of algebraic attacks were already introduced one year earlier in the appendix of the extended version of the original paper [13], under a different name of scenarios $S3_0$ and $S3_1$. There are many other equivalent formulations of the S3 attack, for example instead of annihilators we can talk about absorbing elements: g is the annihilator for f if and only if it is an absorbing element for $f + 1$. We can also speak (as we do a lot in this paper) about algebraic I/O relations, see [11, 10, 16], that generalise the notion of Affine Multiples, known since 1992 [5, 27].

equations of such type exist if and only if either f or $f+1$ have an annihilator of degree $\leq d$, i.e. $\exists g'$ of degree $\leq d$ s.t. $fg' = 0$ or $(f+1)g' = 0$.
Given the existence of one equation of type $zX^d + X^d$ for f (or of an annihilator of degree d), the cipher can be broken with complexity of $\binom{n}{d}^\omega$ as follows. Each equation of the system (#) as follows:

$$z_t = f(\ P\left(L^t\ (s_0, \ldots, s_{n-1})\right)\ )$$

is multiplied by the polynomial $g(\ P\left(L^t\ (s_0, \ldots, s_{n-1})\right)$ and since $fg = h$ it gives an equation of degree d:

$$z_t = h(\ P\left(L^t\ (s_0, \ldots, s_{n-1})\right)\ )$$

Then the system is solved by linearization exactly as described in Section 3.1.
For Sfinks, as we will se later (and as already remarked by the designers of Sfinks [6]) we can have $d = 6$ and the complexity of the attack is about 2^{108}. The keystream required in this attack is $T/4 \approx 2^{36.5}$ bits, as 4 linearly independent equations of type $zX^6 + X^6$ do exist.
Theory vs. practice. It should be noted that in this attack some equations might become linearly dependent, however unlike in [11], this is not a problem at all in algebraic attacks on stream ciphers. This is because the attack method allows to produce as many new equations as may be necessary. In practice, it has been tested for LILI [26] by the authors of [13] in the extended version of this paper. The simulations show clearly that, the number of equations that are not linearly independent in this attack (that could be tolerated with little impact on the complexity) turns out to be really negligible. (See also [2].) Thus the complexity evaluations on algebraic attacks on stream ciphers are expected to be very tight and even rather conservative. It is in fact likely that applying the F5/2 Gröbner bases algorithm [17] to a subset of the resulting equations of degree d, combined with fast linear algebra, should allow to solve the systems with even lower complexity and/or with even less known keystream.

3.3 Fast Algebraic Attacks on Sfinks

Let $D = \binom{n}{d} + \binom{n}{d-1} + \ldots + 1$. Fast algebraic attacks on stream ciphers [14] are based on equations of type $zX^e + X^d$ with $e < d$. They instantiate the scenario S5 according to the terminology of [13]. The principle is as follows: we combine some D consecutive equations from (#) for some D consecutive positions t in such a way that the parts of degree d are eliminated. The equation obtained will be of degree "only" e and will be as follows:

$$\sum_{i=t}^{t+D} \alpha_{t+i} \cdot z_i \cdot g(\ P\left(L^i\ (s_0, \ldots, s_{n-1})\right)\ ) \qquad (*)$$

for some linear combination $(\alpha_0, \ldots, \alpha_{D-1}) \in GF(2)^D$. The same equation applies to each window of D consecutive steps and we will write (and use) it E times, for E overlapping intervals, with $E = \binom{n}{e} + \binom{n}{e-1} + \ldots + 1$. This is because we need to get the final system of degree e that is solvable by linearization (with complexity E^ω).
How to Compute α. The equation above $(*)$ will be true for a proper choice of α, that by definition is such that all the monomials of degree d are eliminated, which can

be written as:

$$0 = \sum_{i=t}^{t+D} \alpha_{t+i} \cdot h(\; P\left(L^i \; (s_0, \ldots, s_{n-1})\right) \;) \qquad (**)$$

In other words, α is a linear combination of D consecutive bits, such that if in our cipher the output Boolean function were h, for any consecutive D steps applying α to bits output by the cipher would give always the sum equal to 0, (i.e. with h instead of f we would have $\sum \alpha_{t+i} z_{t+i} = 0$). Following this observation, in [14] Courtois proposed to use the Berlekamp-Massey algorithm to find this α. This idea has been validated by Armknecht in [2] but later it turned out that there is a much simpler, faster, and more powerful method. Hawkes and Rose, inspired by the invited talk by Massey at FSE 2003 and by an old paper by Key [19], have shown in [18] that it is possible to compute in time of $D \log^2 D$ one single linear combination α that is "universal" for degree up to d, in the sense that it eliminates **any** Boolean function of degree $\leq d$ (and in particular for our function h), see [18]. This is the method we will adopt.
Substitution Step. In [14] it was claimed that the substitution step in the fast algebraic attacks should take about $E \cdot D$ steps. In [18] Hawkes and Rose explain that the simple substitution takes in fact DE^2 operations, and propose an improved FFT-based method that manages (after all) to do it in about $2ED \log D$ operations.

3.4 Summary - Complexity of Fast Algebraic Attacks

Following the work of Courtois, Hawkes and Rose [14, 18], and for any given $(e, d), e < d$ such that there is an equation of type $zX^e + X^d$ for f, we need to perform the following steps in order to perform an algebraic attack on any filter generator such as Sfinks:

0. **Relation Search Step.** Computing the equation[s] of type $zX^e + X^d$ for f. Since we have already handled this step (in few days on a PC) for all interesting cases, we will neglect the complexity of it in this paper (it is most $\binom{17}{d}^\omega$ and can be improved).
1. **Pre-computation Step.** Given the characteristic polynomial of the LFSR, compute the ("universal") α for the degree d. This step requires according to [18] $D \log^2 D$ operations.
2. **Substitution Step.** Write the equations $(*)$ for E consecutive values of t, for example $t = 1, \ldots, E$. This step requires according to [18] about $2ED \log D$ operations.
3. **Solving Step.** Solve these equations by linearization. It requires E^ω operations.

Keystream complexity. The keystream required in the whole attack is $D + E - 1$ which is in general very close to D as if $e < d$ we have $E << D$. We note that if the dimension of the space of I/O relations of type $zX^e + X^d$ for f is $A > 0$, then the keystream can be divided by A. Indeed, in this case, for each window of D output bits, in the substitution step we can generate A equations of degree e. Unfortunately, to use all these, we need to repeat the substitution step A times, which in our best attacks for Sfinks is the dominating step. This would therefore degrade the overall attack complexity (in different cases and for other cryptosystems another step may be dominating, see Table 1 and [18]). Therefore for many attacks we choose to use only one out of these A equations (for some we will give the two evaluations, see the results in Table 1).

3.5 Our Results

In order to apply the fast algebraic attacks we need to search for suitable equations by computer simulations on the Boolean function of Sfinks. Finding such equations allows to execute the fast algebraic attack and there is very little theory[2] that would help to predict whether they do exist for a particular function. One has to check by running a series of simulations. Here are our results:

degree e of g degree d of $h = fg$	0 15	1 11	1 12	2 7	**2** **8**	2 8	3 6	3 7	3 7	3 8	4 6	4 7	5 5	5 6
dimension A of these g out of which we used	1 –	0 –	5 1	0 –	**6** **1**	6 6	0 –	32 1	32 32	136 1	4 1	204 1	0 –	4 1
complexity of the fast algebraic attack [14, 18]:														
pre-computation step	–	–	$2^{79.0}$	–	$\mathbf{2^{59.8}}$	$2^{59.8}$	–	$2^{54.5}$	$2^{54.5}$	$2^{59.8}$	$2^{49.0}$	$2^{54.5}$	–	$2^{49.0}$
substitution step	–	–	$2^{81.4}$	–	$\mathbf{2^{69.7}}$	$2^{72.3}$	–	2^{71}	2^{76}	2^{76}	$2^{71.6}$	$2^{76.9}$	–	$2^{77.3}$
solving step	–	–	$2^{22.5}$	–	$\mathbf{2^{42.1}}$	$2^{42.1}$	–	$2^{60.1}$	$2^{60.1}$	$2^{60.1}$	$2^{76.9}$	$2^{76.9}$	–	$2^{92.8}$
keystream required	–	–	$2^{66.7}$	–	$\mathbf{2^{48.6}}$	2^{46}	–	$2^{43.6}$	$2^{38.6}$	$2^{48.5}$	$2^{38.5}$	$2^{43.6}$	–	$2^{38.5}$

Table 1. Simulations on the dimension of the space of equations of type $zX^e + X^d$ for the Boolean function of Sfinks, i.e. the number of linearly independent functions g of degree $\leq e$ such that $h = fg$ is of degree $\leq d$. The resulting complexity of the fast algebraic attack (following [18]) is computed.

The designers of Sfinks do mention fast algebraic attacks but did not analyse them in due details. Our fastest attack is in the column 6. We need 2^{70} computations and 2^{49} keystream bits. to break Sfinks. The substitution step dominates all the other steps of the attack.

We note also that the attacks in columns 10 or 12 should also break Sfinks slightly faster than the claimed security level of 2^{80}. They require only about $2^{38.5}$ keystream bits, (about 50 Giga-bytes), which can be seen as a realistic attack on Sfinks.

Theory vs. practice. We are about to implement and run a realistic version of this attack.

[2] There is a theory of worse-case attacks that show that some equations always exist for any component of a given size, see [15], however for a specific fixed Boolean function, better equations frequently do exist (e.g. for LILI-128, see [13]). This is maybe because as already suggested by the authors of [13] and [1], one should expect that there are trade-offs between (classical) non-linearity notions, and the resistance against (more recent) algebraic attacks.

4 Summary and Conclusion

Sfinks is not equipped with neither a protection against, nor a sufficient security margin against algebraic attacks on stream ciphers [12–14, 18]. The result is an insecure cipher that can be broken with complexity of about 2^{70} computations and with 2^{49} keystream bits. We also present two attacks slightly faster than 2^{80} that require only 50 Giga-bytes of keystream which is realistic.

Remark. The designers of Sfinks have committed three serious mistakes. First mistake (very, very common one) was to propose a new stream cipher. The second mistake (clearly much less common) was to study known attacks on stream ciphers. Unfortunately here, they made their final mistake: they decided to make it as simple as possible, as elegant as possible, and as fast as possible, resulting in a cipher that is exactly on the edge of being broken, with no extra security. In most areas of mathematics and computer science this would be called neat and elegant. In cryptology it is typically quite foolish to do so. Such ciphers are systematically being broken, it happens again and again...

In the design of block ciphers, Lars Knudsen (and others) have for a long time promoted the following: see how many rounds we need to make it secure, then double it (or even multiply by 4). We believe that stream ciphers should also be designed with such a comfortable security margin. One should not follow the example of Sfinks.

We note that the designers of Sfinks have a good excuse for doing all this: it is very hard to design a cipher with a very small hardware footprint.

Warning: It is hard to see if a cipher is already broken, or not, by already known algebraic attacks. The so called "Algebraic Immunity" of a Boolean function [7] does not help, and it is possible to see that it does **not** give any guaranteed lower-bound on the attack complexity (or only for the simple scenario S3). Equations that lead to better attacks can be found at any moment: following [14] it is not possible to explore systematically all the possibilities offered by the scenario S5 from [13]. The probabilistic versions S4 and S6 proposed in [13] are even more difficult to explore. Thus, better algebraic attacks on Sfinks may remain uncovered. The authors will probably try to repair Sfinks... but even for the next version, there will be no guarantee it is secure.

Open questions: Improve the attack so that it can be handled in practice on a PC. Only the substitution step needs improvement ! See also if doing it A times in parallel is cheaper...

References

1. Frederik Armknecht, Matthias Krause: *Algebraic Atacks on Combiners with Memory,* Crypto 2003, LNCS 2729, pp. 162-176, Springer.
2. Frederik Armknecht: *Improving Fast Algebraic Attacks,* FSE 2004, LNCS, Springer.
3. Steve Babbage: *Cryptanalysis of LILI-128,* Nessie project internal report, available at `https://www.cosic.esat.kuleuven.ac.be/nessie/reports/`, 22 January 2001.
4. Elad Barkan, Eli Biham, and Nathan Keller: *Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication,* In Crypto 2003, LNCS 2729, pp: 600-616, Springer, 2003.
5. I. Blake, X. Gao, R. Mullin, S. Vanstone and T. Yaghoobian, *Applications of Finite Fields*, Kluwer Academic Publishers.
6. An Braeken, Joseph Lano, Nele Mentens, Bart Preneel and Ingrid Verbauwhede: Sfinks specification and source code, April 2005, Available on ECRYPT Stream Cipher Project page, `http://www.ecrypt.eu.org/stream/sfinks.html`

7. Claude Carlet, Will Meier and Enes Pasalic: *Algebraic Attacks and Decomposition of Boolean Functions,* Eurocrypt 2004, pp. 474-491, LNCS 3027, Springer, 2004.
8. A. Clark, E. Dawson, J. Fuller, J. Golic, H-J. Lee, W. Millan, S-J. Moon, and L. Simpson: *The LILI-II Keystream Generator,* Presented at ACISP-2002, the 7th Australasian Conference on Information Security and Privacy. 3 - 5 July 2002. Deakin University, Melbourne, Australia.
9. Don Coppersmith, Shmuel Winograd: *Matrix multiplication via arithmetic progressions,* J. Symbolic Computation (1990), 9, pp. 251-280.
10. Nicolas Courtois: *The security of Hidden Field Equations (HFE)*, Cryptographers' Track Rsa Conference 2001, LNCS 2020, Springer, pp. 266-281.
11. Nicolas Courtois and Josef Pieprzyk: *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations,* Asiacrypt 2002, LNCS 2501, pp.267-287, Springer, A preprint with a different version of the attack is available at `http://eprint.iacr.org/2002/044/`.
12. Nicolas Courtois: *Higher Order Correlation Attacks, XL algorithm and Cryptanalysis of Toyocrypt,* ICISC 2002, November 2002, Seoul, Korea, LNCS 2587, pp. 182-199, Springer. An updated version is available at `http://eprint.iacr.org/2002/087/`.
13. Nicolas Courtois and Willi Meier: *Algebraic Attacks on Stream Ciphers with Linear Feedback,* Eurocrypt 2003, Warsaw, Poland, LNCS, Springer. A long, extended version of this paper is available from `www.nicolascourtois.net`.
14. Nicolas Courtois: *Fast Algebraic Attacks on Stream Ciphers with Linear Feedback,* Crypto 2003, LNCS 2729, pp: 177-194, Springer.
15. Nicolas Courtois: *Algebraic Attacks on Combiners with Memory and Several Outputs,* ICISC 2004, LNCS, to appear in Springer in early 2005. Extended version available on `http://eprint.iacr.org/2003/125/`.
16. Nicolas Courtois: *The Inverse S-box, Non-linear Polynomial Relations and Cryptanalysis of Block Ciphers,* in AES 4 Conference, Bonn May 10-12 2004, LNCS 3373, pp. 170-188, Springer.
17. Jean-Charles Faugère: *A new efficient algorithm for computing Gröbner bases without reduction to zero (F5),* Workshop on Applications of Commutative Algebra, Catania, Italy, 3-6 April 2002, ACM Press.
18. Philip Hawkes, Gregory Rose: *Rewriting Variables: the Complexity of Fast Algebraic Attacks on Stream Ciphers,* in Crypto 2004, LNCS 3152, pp. 390-406, Springer, 2004. Available from `eprint.iacr.org/2004/081/`.
19. Edwin L. Key: *An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators,* IEEE Transactions on Information Theory, Vol IT-22, No. 6, November 1976.
20. J. N. Massey and S. Serconek: *A Fourier Transform Approach to the Linear Complexity of Nonlinearly Filtered Sequences,* In Crypto '94, LNCS 839, Springer-Verlag, pp. 332–340, 1994.
21. Willi Meier and Othmar Staffelbach: *Fast correlation attacks on certain stream ciphers,* Journal of Cryptology, 1(3):159-176, 1989.
22. Willi Meier and Othmar Staffelbach: *Nonlinearity Criteria for Cryptographic Functions,* Eurocrypt'89, LNCS 434, Springer, pp.549-562, 1990.
23. Jacques Patarin: *Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88,* In Crypto'95, Springer, LNCS 963, pp. 248-261, 1995.
24. Alex Biryukov, Adi Shamir: *Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers,* Asiacrypt 2000, LNCS 2248, Springer, pp. 1-13.
25. Adi Shamir, Jacques Patarin, Nicolas Courtois and Alexander Klimov: *Efficient Algorithms for solving Overdefined Systems of Multivariate Polynomial Equations*, Eurocrypt'2000, LNCS 1807, Springer, pp. 392-407.
26. L. Simpson, E. Dawson, J. Golic and W. Millan: *LILI Keystream Generator,* SAC'2000, LNCS 2012, Springer, pp. 248-261, Cf. `www.isrc.qut.edu.au/lili/`.
27. Jacques Patarin: *Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of Asymmetric Algorithms*; Eurocrypt'96, Springer, pp. 33-48.
28. Claude Elwood Shannon: *Communication theory of secrecy systems,* Bell System Technical Journal 28 (1949), see in patricular page 704.
29. Volker Strassen: *Gaussian Elimination is Not Optimal,* Numerische Mathematik, vol 13, pp 354-356, 1969.