SFINKS: A Synchronous Stream Cipher for Restricted Hardware Environments [⋆]

An Braeken[⋆⋆] and Joseph Lano[⋆⋆⋆] and Nele Mentens and Bart Preneel and Ingrid Verbauwhede

Katholieke Universiteit Leuven
Dept. Elect. Eng.-ESAT/SCD-COSIC,
Kasteelpark Arenberg 10, 3001 Heverlee, Belgium
{an.braeken,joseph.lano,nele.mentens,bart.preneel,ingrid.verbauwhede}@esat.kuleuven.be

Abstract. We present SFINKS, a low-cost synchronous stream cipher for hardware applications with an associated authentication mechanism. The stream cipher is based on a Simple Filter generator, using the INverse function in $\mathbb{F}_{2^{16}}$ to generate the Key Stream. The design is based on simple and well-studied concepts, and its security is analyzed with respect to the portfolio of known cryptanalytic attacks for filter generators.

1 Introduction

For efficient encryption of data, cryptography mainly uses two types of symmetric algorithms, block ciphers and stream ciphers. In the past decades, block ciphers have become the most widely used technology. This is mainly due to the block cipher standard DES [32] and its successor AES [33]. The current AES is a secure encryption algorithm that offers excellent performance on a variety of hardware and software environments.

As block ciphers are often used in a stream cipher mode such as CTR and OFB, stream ciphers may offer equivalent security at a lower cost. The aim of this paper is to propose a low-cost synchronous stream cipher for hardware applications with an associated authentication mechanism. The design we propose is a simple synchronous stream cipher using a memoryless nonlinear filter. We will motivate our choices made for the building blocks in the following sections.

The outline of this paper is as follows. First, we introduce some preliminary concepts in Sect. 2. In Sect. 3, we give the objectives of our design. In Sect. 4, a description of the entire design is given. We discuss the security of our design with respect to the various cryptanalytic attacks in Sect. 5, motivating the design choices made. Perfomance and implementation aspects of the design are addressed in Sect. 6 and Sect. 7 respectively. We present possible tweaks of the design in Sect. 8 and conclude in Sect. 9.

[⋆] This work was supported in part by the Concerted Research Action (GOA) Mefisto 2000/06 of the Flemish Government, by FWO projects and by the European Commission through the IST Programme under Contract IST2002507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

[⋆⋆] F.W.O. Research Assistant, sponsored by the Fund for Scientific Research - Flanders (Belgium)

[⋆⋆⋆] Research financed by a Ph.D. grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen)

2 Preliminaries

In this section, we list some properties of the two building blocks that are used in a nonlinear filter generator, namely an LFSR and a Boolean function. For a more thorough treatment we refer to [19, 37].

2.1 Linear Feedback Shift Registers

Definition 1 *A* Linear Feedback Shift Register (LFSR) *of length L is a collection of L 1-bit memory elements $s_t^0, s_t^1, \ldots, s_t^{L-1}$. At each time t the memory is updated as follows:*

$$\begin{cases} s_t^i = s_{t-1}^{i+1} \text{ for } i = 0, \ldots, L-2 \\ s_t^{L-1} = \bigoplus_{i=1}^{L} c_i \cdot s_{t-1}^{L-i} \,. \end{cases} \tag{1}$$

where the c_i are fixed binary coefficients that define the feedback equation of the LFSR. The LFSR stream $(s_t)_{t\geq 0}$ consists of the successive values in the memory element s_0.

Associated with an L-bit LFSR is its *feedback polynomial $P(X)$* of degree L, $P(X) = 1 + \sum_{i=1}^{L} c_i \cdot X^i$. The *weight* of the feedback polynomial is equal to the number of its nonzero terms. In practical designs, a feedback polynomial is chosen to be *primitive*. This implies that every nonzero initial state produces an output sequence with maximal period $2^L - 1$, which is also called a pn-sequence.

For many cryptanalytic attacks, it is useful to search low-weight multiples of the feedback polynomial, see [6, 20, 16]. The number $m(D, w)$ of multiples $Q(X) = 1 + \sum_{i=1}^{D} c_i \cdot X^i$ of the polynomial $P(X)$, with degree less or equal than D and weight w, can be approximated by [6]:

$$m(D, w) \approx \frac{D^{w-1}}{(w-1)! \cdot 2^L} \,. \tag{2}$$

It is interesting to know from which D_{min} we can expect a first multiple $Q(X)$ of weight w to start appearing. It follows from (2) that:

$$D_{min}(w) \approx (2^L \cdot (w-1)!)^{\frac{1}{w-1}} \,. \tag{3}$$

The most efficient approach, known to date, to search for these low-weight multiples is a birthday-like approach, see [16]. The precomputation complexity P needed to find all multiples $Q(X)$ of weight w and degree at most D can be approximated by:

$$P(D, w) \approx \frac{D^{\lceil \frac{w-1}{2} \rceil}}{\lceil \frac{w-1}{2} \rceil !} \,. \tag{4}$$

2.2 Boolean Functions

A *Boolean function* f is a mapping from $\mathbb{F}_2^{\varphi}$ into $\mathbb{F}_2$. The *support* of f is defined as $\sup(f) = \{\overline{x} \in \mathbb{F}_2^{\varphi} : f(\overline{x}) = 1\}$. The cardinality of $\sup(f)$ represents the weight $\mathrm{wt}(f)$ of the function.

A Boolean function can be uniquely represented by means of its *algebraic normal form (ANF)*:

$$f(\overline{x}) = f(x_{\varphi-1}, \ldots x_0) = \bigoplus_{(a_{\varphi-1}, \ldots, a_0) \in \mathbb{F}_2^{\varphi}} h(a_{\varphi-1}, \ldots, a_0)\, x_{\varphi-1}^{a_{\varphi-1}} \ldots x_0^{a_0} \,, \tag{5}$$

where f and h are Boolean functions on $\mathbb{F}_2^{\varphi}$. The *algebraic degree* of f, denoted by $\deg(f)$, is defined as the highest number of variables in the terms $x_{\varphi-1}^{a_{\varphi-1}} \dots x_0^{a_0}$ in the ANF of f.

Alternatively, a Boolean function can be represented by its *Walsh spectrum*:

$$W_f(\overline{\omega}) = \sum_{\overline{x} \in \mathbb{F}_2^{\varphi}} (-1)^{f(\overline{x}) \oplus \overline{x} \cdot \overline{\omega}} = 2^{\varphi-1} - 2\,\mathrm{wt}(f \oplus \overline{x} \cdot \overline{\omega}), \tag{6}$$

where $\overline{x} \cdot \overline{\omega} = x_0\omega_0 \oplus x_1\omega_1 \oplus \dots \oplus x_{\varphi-1}\omega_{\varphi-1}$ is the *dot product* of $\overline{x}$ and $\overline{\omega}$.

Several properties are of importance for Boolean functions from a cryptographic viewpoint. A function is said to be *balanced* if $\mathrm{wt}(f) = 2^{\varphi-1}$. The *nonlinearity* N_f of the function f is defined as the minimum distance between f and any affine function; it can be calculated as $N_f = 2^{\varphi-1} - \frac{1}{2}\max_{\overline{\omega} \in \mathbb{F}_2^n} |W_f(\overline{\omega})|$. The *best affine approximation* $l(\overline{x})$ is associated with this notion. We will say that f has *bias* ϵ if it has the same output as its best affine approximation with probability $0.5 + \epsilon$. It is easy to see that $\epsilon = N_f/2^{\varphi} - 0.5 = \frac{\max_{\overline{\omega} \in \mathbb{F}_2^n} |W_f(\overline{\omega})|}{2^{\varphi+1}}$. A function f is said to be correlation-immune [38] of order ρ, $CI(\rho)$, if and only if its Walsh transform W_f satisfies $W_f(\overline{\omega}) = 0$, for $1 \leq \mathrm{wt}(\overline{\omega}) \leq \rho$. If the function is also balanced, then the function is called t-resilient.

The lowest degree of the function g from $\mathbb{F}_2^{\varphi}$ into $\mathbb{F}_2$ for which $f \cdot g = 0$ or $(f+1) \cdot g = 0$ is called the *algebraic immunity (AI)* of the function f [29]. The function g is said to be an *annihilator* of f if $f \cdot g = 0$.

A vectorial Boolean function F from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$, also called (n, m) S-box, can be represented by an m-tuple $(f^{m-1}, \dots, f^0)$ of Boolean functions f^i on $\mathbb{F}_2^n$ (corresponding to the output bits).

3 Design Goals

Easy Analysis. Most research on stream ciphers has been focused on synchronous regularly clocked LFSR-based stream ciphers, resulting in a number of generic cryptanalytic attacks on these designs, such as (fast) correlation attacks [38, 30] and, recently, (fast) algebraic attacks [9, 8]. At the SASC workshop, it was suggested that these attacks make it very hard to design a secure LFSR-based stream cipher. However, we believe that LFSR-based designs are, despite the possibilities of long key stream attacks, a good choice for hardware applications, provided we can prove the resistance of the design to a whole set of cryptanalytic attacks. The study of such designs can be brought back to the study of its components: the LFSR and the Boolean function. In this paper, we will provide a framework that relates the security of the cipher with respect to various cryptanalytic attacks (distinguishing attacks, correlation attacks, algebraic attacks, resynchronization attacks. . .) to a few important parameters: the feedback taps of the LFSR, the input taps to the Boolean function, and the properties of the Boolean function (degree, algebraic immunity, Walsh spectrum). We will then search a Boolean function that has all the required properties and is easy to implement in hardware.

Performance. The stream cipher is designed for use in restricted hardware environments. LFSRs are very efficient in hardware, and are the preferred choice in many fielded applications such as A5/1 in GSM and E0 in Bluetooth (we will not comment on the security of these designs, but note that these designs were implemented without any public evaluation taking place). These designs have an internal state of 64 bits and 128 bits respectively, so our choice for a 256-bit internal state is a logical next step. The building blocks of our design are chosen to be easily implementable in hardware, since we want a design with a low gate count that still achieves a reasonable throughput.

4 Description of the Design

Our design is a synchronous stream cipher based on a nonlinear filtered key stream generator with an associated authentication mechanism. The layout of our design is presented in Fig. 1. The internal state consists of an LFSR of 256 bits. Our Boolean function is the 16-bit inversion S-box from which we take one bit as output after linear masking. We also use an output bit of the inversion function for the generation of the 64-bit MAC. We now discuss the different building blocks of the algorithm. The explanations given here should be sufficient to implement SFINKS. For a complete and unambiguous description, we refer to our C code.

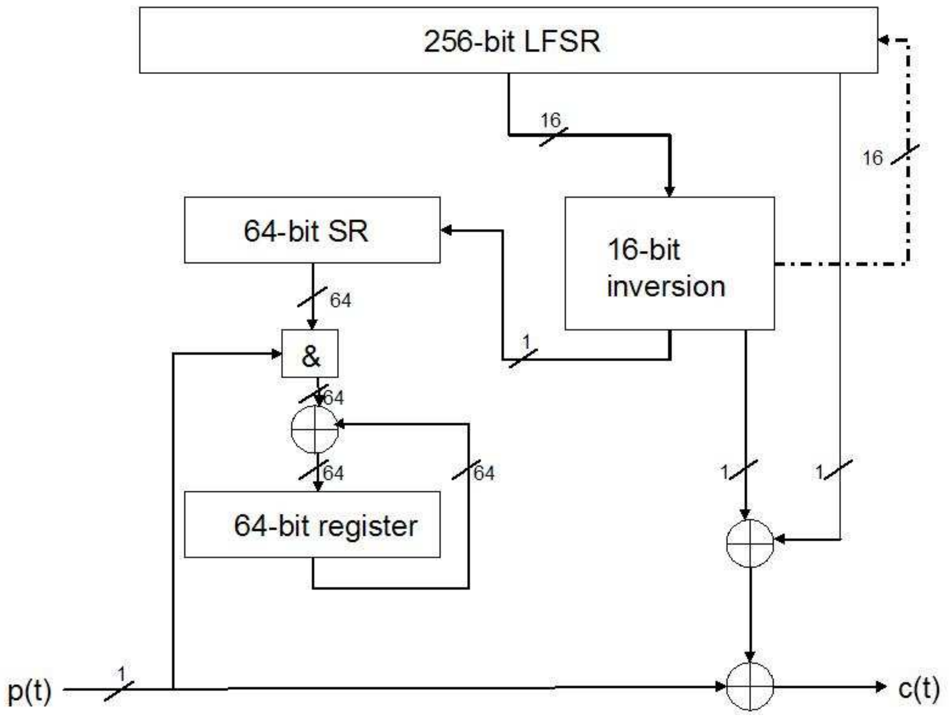

Fig. 1. Layout of the stream cipher

4.1 Key, IV and MAC Length

Our algorithm is aimed at offering medium-term security, which is reflected in the length of 80 bits of the secret key $K = k_{79} \dots k_0$. Also, the length of our initialization vector $IV = iv_{79} \dots iv_0$ is 80 bits. For the MAC, we choose a length of 64 bits. If this would be required, we can easily adapt these parameters in a straightforward way.

4.2 LFSR

An LFSR of length $L = 256$ bits is used. It is a Fibonacci LFSR with the following weight 17 feedback polynomial:

$$P(X) = 1 + X^{44} + X^{62} + X^{64} + X^{69} + X^{93} + X^{105} + X^{131} + X^{141} + X^{149} + X^{171} + X^{190} + X^{192} + X^{204} + X^{208} + X^{242} + X^{256}\,. \tag{7}$$

In other words, the recursion formula for the LFSR stream is as follows:

$$s_{t+256} = s_{t+212} \oplus s_{t+194} \oplus s_{t+192} \oplus s_{t+187} \oplus s_{t+163} \oplus s_{t+151} \oplus s_{t+125} \oplus s_{t+115} \oplus s_{t+107} \oplus s_{t+85} \oplus s_{t+66} \oplus s_{t+64} \oplus s_{t+52} \oplus s_{t+48} \oplus s_{t+14} \oplus s_t\,. \tag{8}$$

4.3 Filter Function

The nonlinear filter function used for generating the key stream is a Boolean function f from $\mathbb{F}_2^{17}$ into $\mathbb{F}_2$. At each time t, 17 bits are taken from the LFSR, put into the word $\overline{x}_t \in \mathbb{F}_2^{17}$ and input to the Boolean function to calculate the key stream z_t as follows:

$$z_t = f(\overline{x}_t) = f(x_t^{16}, \ldots, x_t^0) = (INV(x_t^{16}, \ldots, x_t^1)\&1) \oplus x_t^0\,. \tag{9}$$

In this formula, the input word $\overline{x}_t$ is selected as follows from the LFSR stream:

$$\overline{x}_t = (s_{t+255}, s_{t+244}, s_{t+227}, s_{t+193}, s_{t+161}, s_{t+134}, s_{t+105}, s_{t+98}, s_{t+74}, s_{t+58}, s_{t+44}, s_{t+21}, s_{t+19}, s_{t+9}, s_{t+6}, s_{t+1}, s_t)\,. \tag{10}$$

The main building block is the vectorial Boolean function $INV()$ from $\mathbb{F}_2^{16}$ into $\mathbb{F}_2^{16}$. $INV()$ calculates the inverse of the input in the field $\mathbb{F}_{2^{16}}$ modulo the primitive polynomial $X^{16} + X^5 + X^3 + X^2 + 1$. We denote the output of this vectorial Boolean function by $\overline{y}_t \in \mathbb{F}_2^{16}$. Note that we will only use one bit of this inverse S-box for our key stream generation.

Some important cryptographic properties of the filter function on $\mathbb{F}_2^{17}$, which we will address into more detail in the following sections, are as follows: the function is balanced, the degree of the function is 15, the nonlinearity is 65024, its best affine approximation has bias $\epsilon = 2^{-8}$ and its algebraic immunity is equal to 6.

4.4 Resynchronization Mechanism

We first set all internal state bits to zero and start our algorithm at $t = -128$. First K and IV are loaded into the $LFSR$. We set $s_{-128}^{255} = iv_{79}, \ldots, s_{-128}^{176} = iv_0$, $s_{-128}^{175} = k_{79}, \ldots, s_{-128}^{96} = k_0$. and $s_{-128}^{95} = 1$. We also set $\overline{y}_{-134}$ to $\overline{y}_{-128}$ equal to zero (We will see later that this amounts to clearing the pipeline stages in HW, if we choose to use pipelining for increasing the throughput).

We then start 128 resynchronization steps, for $t = -127, \ldots -1, 0$, as follows. The LFSR is clocked in a special way, where 16 bits of the LFSR are also XORed with bits from the inversion function as follows:

$$s_t^j = s_{t-1}^{j+1} \oplus \overline{y}_{t-7}^{j \bmod 16}\,, \tag{11}$$

for the 16 values of $j = \{11, 17, 41, 52, 66, 80, 111, 118, 142, 154, 173, 179, 204, 213, 232, 247\}$.

At each time, we then need to calculate $\overline{y}_t$, as we will use these 16 bits to be fed back into the LFSR as in the equation above. We will also shift the bit y_t^1 into the shift register for MAC generation as during normal operation, and always XOR the register contents into the MAC state, see below. At $t = 0$, the resynchronization is finished and we can start the key stream generation.

4.5 MAC Algorithm

Our MAC generation algorithm is based on concepts similar to [26]. If no authentication method is required, SFINKS can also be implemented without this authentication method.

For MAC generation, we use the shift register $R_t = r_t^{63} \dots r_t^0$. During the frame encryption, the value of the MAC will be accumulated in the register $M_t = m_t^{63} \dots m_0^{63}$ which is set to zero at the start of each frame. For all times $t = -127, \dots framelength$, the shift register is updated as follows:

$$\begin{cases} r_t^i = r_{t-1}^{i+1} \text{ for } i = 0, \dots, 62 \\ r_t^{63} = \overline{y}_t^1 \, . \end{cases} \tag{12}$$

For every plaintext bit $p_t, t = 1, \dots framelength$, we update the MAC state by $M_t = M_{t-1} \oplus p_t \cdot R_{t-1}$[1]. At the end of the frame, the 64 bits of M are sent to the receiver, after they have been encrypted during 64 final clocks of the key stream generator.

5 Security Analysis

In this section we address the security of SFINKS with respect to the various existing cryptanalytic attacks against LFSR-based stream ciphers. We explain why our choice of building blocks should provide a good resistance against these attacks.

5.1 Berlekamp-Massey Attacks

For a Boolean function of degree d, the linear complexity LC of the resulting key stream is upperbounded by $\sum_{i=0}^{d} \binom{L}{i}$. Moreover, it is very likely that the LC of the key stream is lowerbounded by $\binom{L}{d}$ and that its period remains equal to $2^L - 1$. We refer to [37, 23] for more details. The Berlekamp-Massey attack requires $2 \cdot LC$ data and has a complexity of LC^2. Using the parameters $L = 256, d = 15$, this attack is clearly of no concern for the design.

5.2 Tradeoff Attacks

Time-Memory-Data Tradeoff attacks [3, 18, 5] are generic attacks against stream ciphers. To prevent these attacks, the internal state should be at least twice the key size. Our design satisfies this requirement, as the internal state of 256 bit is far bigger than the 80-bit secret key.

Recently, it was noticed that a tradeoff attack can also be mounted directly on the secret key, irrespective of the internal state size [21]. To prevent this attack, we choose the size of the initialization vector to be 80 bits, the same size as the key [11]. Note that it is the responsibility of the implementer to make sure that the initial IV is chosen in a correct, non-fixed way, to prevent these tradeoff attacks.

[1] note that, as mentioned in the resynchonization mechanism, the MAC is updated by $M_t = M_{t-1} \oplus R_{t-1}$ during the resynchronization steps

5.3 Distinguishing Attacks

The distinguishing attack we describe here is based on the framework developed in [12] combined with the mathematical results from [31].

The idea of the attack is the following. The LFSR stream has very good statistical properties, but of course there are linear relations, which are determined by the feedback polynomial $P(X)$ and its multiples.

Given a linear relation of weight w in the LFSR stream, the same relation for the corresponding bits of the key stream will hold with some probability different from one half, because the Boolean function f does not destroy all linear properties. It is shown in [31] that this bias is equal to:

$$\varepsilon' = \frac{\sum_{\overline{\omega}=0}^{2^{\varphi}-1} (W_f(\overline{\omega}))^w}{2^{\varphi \cdot w+1}} \tag{13}$$

It is important to notice that the higher the weight of the linear relation, the smaller the bias will be. An even distribution of the Walsh spectrum of the Boolean function will also decrease the bias.

To distinguish the key stream from random, the number of samples needed is in the order of $\frac{1}{\varepsilon'^2}$. The time complexity of the attack is in the same order, the data complexity can be decreased further to some extent by using multiple linear relations simultaneously.

The Boolean function based on inversion is a good choice with respect to this attack. Its nonlinearity is close to the best known for balanced functions, and its Walsh spectrum is spread rather evenly. This is for instance not the case for another class of functions that have high nonlinearity, the plateaued functions [40], where the Walsh spectrum is three-valued. One can easily check that these functions behave worse with respect to this distinguishing attack.

The attack hence proceeds as follows: for every weight w, starting from $w = 3$, we know that we will have a linear recursion with degree around D_{min} as given by (3). We can then precompute this linear recursion $Q(X)$ with complexity given by (4). In the actual attack, we use this low-weight relation $Q(X)$ to distinguish the key stream from a random sequence. To do so, we need around $D_{min} + \frac{1}{\varepsilon'^2}$ data and time.

Note that if we would choose a feedback polynomial of very low weight (e.g., a trinomial), the attack would be easier to mount as no precomputation is needed and the bound on D_{min} does not hold. Our choice of feedback polynomial, with weight 17, does not have problems in this respect. The complexities of the distinguishing attack on SFINKS are given in Table 1. All numbers in the table are given as base 2 logarithms.

In [12], it is shown that this distinguishing attack can be turned into a key recovery attack only if many weight three multiples are found. Therefore, the knowledge of around $2^{L/2}$ (see (2)) bits leads to an efficient attack for any filter generator with internal state of size L.

There has been some debate on the relevance of long key stream distinguishing attacks during the NESSIE stream cipher competition [36]. Whereas the time complexity is a matter of resources on the attacker's side, the amount of key stream available can be limited on the encryption side to a practical amount. We suggest limiting the maximum amount of key stream generated from a K/IV pair to a practical limit of 2^{40} bits. This will make it impossible to exploit any multiples with

Table 1. Logarithm of complexities of the distinguishing attacks

Weight	Precomputation	D_{min}	bias ε'	Attack complexity
3	128.50	128.50	-15.00	128.50
4	171.39	86.19	-16.00	86.19
5	129.29	65.15	-29.32	65.15
6	155.16	52.58	-30.68	61.36
7	130.16	44.25	-43.60	87.20
8	148.72	38.33	-45.19	90.38
9	131.06	33.91	-57.84	115.67
10	145.58	30.50	-59.61	119.21

weight less than 8, as their degree will most likely be higher than 2^{40}. This makes it impossible for an attacker to find any bias in the key stream.

5.4 Affine Approximation Attacks (correlation like)

Correlation and fast correlation attacks exploit the correlation between the LFSR stream s_t and the key stream z_t. These attacks can be seen as a decoding problem, since the key stream z_t can be considered as the transmitted LFSR stream s_t through a binary symmetric channel with error probability $p = P_{t\geq 0}(z_t \neq s_t)$. For the nonlinear filter generator, p is determined by $0.5 + \epsilon$. Therefore, the attack consists of a fast decoding method for any LFSR code C of length N (the amount of available key stream) and dimension L (the length of the LFSR), where the length N of the code is lowerbounded by Shannon's channel coding theorem:

$$N \geq \frac{L}{C(p)} = \frac{L}{1 + p\log_2 p + (1-p)\log_2(1-p)} \approx \frac{\ln(2)L}{2\epsilon^2}\,. \tag{14}$$

With the parameters $L = 256$, $\epsilon = 2^{-8}$, we obtain that the number of required key stream bits is at least $N = 2^{22.47}$. Consequently, in theory there should exist a decoding algorithm with zero error probability for this configuration of parameters, but in practice no efficient general decoding algorithm is known for achieving the channel capacity.

Besides the maximum-likelihood (ML) decoding, which has very high complexity of $L \cdot 2^L$, mainly two different approaches have been proposed in the literature. In the first approach, the existence of sparse parity check equations for the LFSR code are exploited. These parity check equations correspond with the low weight multiples of the connection polynomial. In this way, the LFSR code can be seen as a low-density parity-check (LDPC) code and has several efficient iterative decoding algorithms. In the second approach, a smaller linear $[n, l]$ code with $l < L$ and $n > N$ is associated to the LFSR on which ML decoding is performed. The complexity of both approaches highly depends on the existence of low-degree multiples. As shown above, the precomputation time for finding these low-degree multiples is very high. Moreover, the approach requires long key streams and suffers from a high decoding complexity. Therefore, we can conclude that due to the large length of the LFSR and the good nonlinearity of the filter function, all approaches studied in literature so far do not succeed. To give an idea of the complexity of these attacks, we concentrate on the attack of Canteaut and Trabbia [6]. When parity-check equations with weight

w are used, the required key stream N is approximated by:

$$\left(\frac{1}{2\epsilon}\right)^{\frac{2(w-2)}{w-1}} 2^{\frac{L}{w-1}} , \tag{15}$$

and the complexity for the attack can be roughly estimated by

$$\left(\frac{1}{2\epsilon}\right)^{\frac{2w(w-2)}{w-1}} 2^{\frac{L}{w-1}} . \tag{16}$$

Table 2 shows the numerical values for the attack on our design, which turn out to be impractical. All data in the table are base 2 logarithms.

Table 2. Logarithm of complexities of the correlation attack

Weight	Precomputation	Data complexity	Decoding complexity
3	135.00	135.00	149.00
4	188.33	94.66	122.67
5	148.00	74.50	116.50
6	184.61	62.40	118.40
7	160.41	54.33	124.33
8	189.70	48.57	132.57
9	172.42	44.25	172.42
10	197.54	40.89	197.54

5.5 Algebraic Attacks

In algebraic attacks [9], a system of nonlinear equations between input and output is constructed and subsequently solved. The complexity of solving this system of equations is highly dependent on the degree of these equations. In the usual algebraic attack, equations between one bit of the output of the filter generator and the initial state of the LFSR are searched. These equations are then solved by linearization. The lowest possible degree d, also called the AI, for these equations is obtained by the annihilators of the filter function and its complement. The total complexity $C(L, d)$ of the algebraic attack on a stream cipher with a linear state of L bits and equations of degree d is then determined by

$$C(L,d) = \left(\sum_{i=0}^{d} \binom{L}{i}\right)^{\omega} \approx L^{\omega \cdot d} , \tag{17}$$

where ω corresponds to the coefficient of the most efficient solution method for the linear system. We use here Strassen's exponent [39] which is $\omega = log_2(7) \approx 2.807$. Note that in the complexity analysis, the linearization method is used for solving the equations. It is an open question if other algorithms like the Buchberger algorithm, F4 or F5 [13] can significantly improve this complexity. Also, the total number of terms of degree less than or equal to d is considered in the complexity, while in general nothing is known about the proportion of monomials of degree d that appear in the system of equations. Therefore, a sufficient security margin should be taken into account.

Table 3 shows the numerical values for the algebraic attack on an LFSR of length 256 and AI between 4 and 8. All data in the table are base 2 logarithms. The algebraic immunity of a Boolean function can be calculated using an algorithm described in [29]. We have computed the algebraic immunity of our Boolean function and found that it is equal to 6 (exactly 4 annihilators of f with degree 6). This leads to an attack with complexity approximately equal to $2^{107.97}$ and is quite close to the edge.

Table 3. Logarithm of complexities of the algebraic attack

AI	Data complexity	Attack complexity
4	27.40	76.92
5	33.07	92.84
6	38.46	107.97
7	43.62	112.46
8	48.59	136.41

In the fast algebraic attacks [8], the attacker tries to decrease the degree d of the system of equations even further by searching for relations between the initial state of the LFSR and several bits of the output function simultaneously. The equations where the highest degree terms do not involve the key stream bits are considered. In an additional precomputation step, linear combinations of these equations are searched which cancel out these highest degree terms in order to obtain equations of degree e. It is clear that the complexity for solving these equations is now $C(L, e)$.

For our function, a reduction of the degree of the equations to 4 should be possible to obtain a certificational attack on the design. The framework for algebraic attacks is for the moment not entirely clear. We believe it is important to further investigate the properties of this and other Boolean functions with respect to fast algebraic attacks, and to see how we could improve the algebraic immunity of the Boolean function while keeping its other desirable properties.

5.6 Resynchronization Attacks

A framework for resynchronization attacks has been given in [10, 2]. Some examples of concrete attacks can be found in [28, 22]. The aim of these attacks is mainly to find relations between the key stream bits of many frames and the secret key loaded into the internal state during resynchronization.

In order to prevent these attacks, it has been shown that one needs to have a resynchronization mechanism that adds enough confusion and diffusion to the initial state. In other words, every bit of the initial state should depend on a large number of key bits and IV bits, and the relation that subsist between the initial state and the key/IV should be highly nonlinear and have a nondetectable bias. We refer to [2] for more details.

Simulations and mathematical arguments suggest that the proposed resynchronization mechanism satisfies these requirements. By the choice of the LFSR taps, filter taps and feedback taps, the nonlinearity of the filter function will be well integrated into the initial state and therefore lead to the desired diffusion and avalanche properties. The successive applications of the 16-bit inversion function during the resynchronization provides confusion and ensures that all remaining biases have negligible probability.

5.7 Other Attacks

Our design satisfies the properties, suggested by Golic in [17], for resistance against inversion attacks. The distance between the first and the last output tap is maximal to prevent guessing a subset of the state first. Our 17 input taps to the output function are also a full positive difference set, which ensures that if two bits enter the output function at time t, they will never enter the output function simultaneously at any other time. We have also chosen our output function of the form $g(x_{16}, \ldots, x_1) \oplus x_0$, which produces a purely random sequence provided the input sequence is random. Consequently, the filter function also satisfies the property of 1-resiliency [38].

These choices ensure that the design has very little structural weaknesses that the attacker can exploit. They seem, combined with the large internal state, to provide adequate security against other attacks that exploit the structure of the design, such as attacks using decimation of the key stream [17, 14], guess and determine attacks and BDD-based cryptanalysis [25].

5.8 Security of the MAC Algorithm

We give here a brief sketch of the security of the MAC algorithm. We refer to [26] for a thorough treatment and proofs for similar situations.

Let $\mathbf{p}$ be a plaintext of length t (denoted as a $t \times 1$ column vector), and let $M = MAC_{K,IV}(\mathbf{p})$. An attacker succeeds in breaking the authentication if he can find $\mathbf{p}'$ (of length t') and M' such that $M' = MAC_{K,IV}(\mathbf{p}')$.

To see why this will not work, we describe our MAC algorithm as a matrix multiplication:

$$M = A \cdot \mathbf{p} + \mathbf{z}\,, \tag{18}$$

where $A = (R_1, R_2, \ldots R_t)$ is a Toeplitz matrix[2] and $\mathbf{z} = (z_{t+1} \ldots z_{t+64})^T$.

As usual, we are in a known plaintext scenario. We suppose the attacker knows $\mathbf{p}$ (and hence $z_1 \ldots z_t$) and also M. If our resynchronization mechanism is secure, the matrix A will contain enough randomness and the attacker will not be able to know anything about this matrix. Hence, any modification to $\mathbf{p}'$ will change M to M' in a pseudorandom way, which the attacker can not control without knowing the secret key (which he does not know if our key stream generator is secure). It thus follows that our MAC is secure if both the resynchronization mechanism and the key stream generation are secure.

Vice versa, the MAC algorithm does not lower the security of the stream cipher. By looking at the MAC, the attacker does not learn any extra information, as before outputting the MAC we have first XORed it with 64 bits of new key stream, which has not been used to encrypt any plaintext and hence is not known by an attacker in a known plaintext scenario.

6 Hardware Performance

The hardware cost of an algorithm is measured in NAND gates. One NAND gate is considered to have a unit area in CMOS standard cell based hardware. We will

[2] a matrix for which all elements on the same diagonal are equal

now list the main hardware costs of our design. We use the figures from [4] for the number of gates needed. Note that these figures are conservative upper bounds based on standard cells. Using specialized libraries may lower the number of gates needed.

When trying to minimize the number of gates, a first constraint we run into is the size of the internal state of 256 bits. Further, the complexity of the implementation of the Boolean function and the MAC mechanism have to be added.

The LFSR requires 256 FF (flip-flops) for the 256 bits, which is approximately 3072 gates. For the LFSR recursion we need 15 XOR gates and we need 16 XOR gates for the feedback during resynchronization, which requires 77.5 more gates.

The design of the inverse function in $\mathbb{F}_{2^{16}}$ is based on an approach similar to the one described in [35] by working in composite fields. The main idea is to decompose the inversion in $\mathbb{F}_{2^{16}}$ into operations in the field $\mathbb{F}_{2^2}$. We refer to Appendix A for a more detailed analysis of this approach. This implementation requires 398 XORs gates and 75 AND gates, and thus consists of about 1107.5 NAND gates. In order to increase the clock frequency, we added registers leading to 7 pipeline stages. In total, we need 84 FFs for the pipelining, bringing the total number of NAND gates for the Boolean function to 2115.5.

Further we need two more XOR gates for the linear masking and the addition of the key stream to the plaintext. For the MAC algorithm, two registers of 64 bits, 64 XOR gates and 64 AND gates are required. This consists of around 1792 gates.

The total gate count for this design, which is already pipelined and has an incorporated MAC algorithm, can be approximated by 7062 gates. To further reduce the gate count if needed, some possibilities could be reducing the MAC length to 32 bits, leaving out the pipelining, searching for a more efficient implementation for the inversion function and using specialized libraries.

We have implemented the design on a Xilinx Spartan xc3s5000 FPGA, and we achieve a clock frequency of 172 MHz (at each clock one bit of ciphertext is generated). We are currently implementing some versions of the design for achieving various tradeoffs between gate count and encryption speed.

7 Implementation Issues

An encryption algorithm needs to be mathematically secure, but it must also be implementable it in a secure way. Many side-channel attacks on encryption algorithms have been developed in recent years. The most notable class of attacks are differential power analysis (DPA) attacks [24] and related attacks. Any design in which operations, depending on a subset of the key bits, are performed repeatedly is potentially vulnerable to this class of attacks. They hence also apply to stream ciphers with resynchronization mechanism, as shown in [27]. We believe that especially irregularly clocked designs are vulnerable for these side-channel attacks, which is one of the reasons for choosing a regularly clocked cipher.

A possible countermeasure against DPA and related attacks for block ciphers is a masking scheme, as introduced in [1]. SFINKS has the particularity that it can be easily masked with similar concepts, as opposed to other stream ciphers where we know of no straightforward way of using a masking scheme. The linear parts can be masked in a simple way and the inversion function can be masked in a way similar to the approach used for masking the AES S-box.

We now roughly explain how to mask our design. The first component of the cipher is the LFSR which is a linear transformation and straightforward to mask.

The second component consists of the filter function where the nonlinear part is the inverse function in $\mathbb{F}_{2^{16}}$. As shown in [34], in order to mask the inverse function, the computations should be performed in $\mathbb{F}_{2^2}$, since the inverse in this field is a linear transformation and thus easy to mask. Extending the masking scheme of [34] to one layer higher, leads to a masking scheme for the inverse in $\mathbb{F}_{2^{16}}$. We refer to the extended version for a detailed analysis of the masking scheme.

8 Tweaks

We have proposed a clear design and discussed its security against all known attacks. The filter function in the design is based on the inverse S-box in $\mathbb{F}_2^{16}$. Since we only use 2 output bits of this function (one for the key stream and one for the authentication), there is still a lot of freedom in the design.

The first possibility is to increase the throughput. Instead of outputting 1 bit, we can output more bits. Therefore, we studied all linear and nonlinear combinations of 2 and 3 output bits of the inverse S-box in $\mathbb{F}_2^{16}$. All of them maintain the AI of 6 and have degree 15. However, the best bias for the affine approximation decreases to $\epsilon = 2^{-7.52}$ and $\epsilon = 2^{-7.11}$ for 2 respectively 3 output bits. From the analysis above, we may conclude that this is not an immediate security concern. When outputting 4 bits, there exist nonlinear combinations which have AI equal to 5. However, further analysis is required for investigating if the extra information that is leaked in this way can lead to more efficient attacks. For instance, is it possible to exploit the fact that all output functions of the inverse S-box are affine equivalent [15]?

Another possibility, which makes the analysis harder but may increase the security, is to destroy the linearity of the state. We could consider a filter generator with memory by simply feeding some remaining bits from the inversion function into a nonlinear memory. Another possibility is to feedback bits of the inverse S-box into the LFSR during key stream generation, in a manner similar to what is done during resynchronization. In both cases, it seems that the added nonlinearity may allow us to increase the throughput.

Clearly, the analysis of both designs is much more difficult. Although we expect to obtain a higher security since the nonlinearity of the filter function is now reused for destroying the linearity in the first part of the generator, we recommend to obtain first a more thorough analysis for these schemes.

Since we are aware of the fact that the security against the (fast) algebraic attack is really on the edge due to the low AI of the inverse function in $\varphi = 16$, we propose some other filter functions with strictly higher AI, comparable nonlinearity, which are still good with respect to the hardware complexity.

- The inverse function in $\mathbb{F}_2^{17}$ has degree 16, AI equal to 7, and $\epsilon = 2^{-8.50}$. Moreover, the function can be easily implemented by performing the computations in the normal basis $\{\alpha, \alpha^2, \alpha^4, \cdots, \alpha^{2^{16}}\}$ of $\mathbb{F}_2^{17}$. Computing the power function in this basis will not change the properties of nonlinearity, degree, AI and Walsh spectrum of the output functions, since power functions in different bases are linearly equivalent. Squaring in this basis represents simply a cyclic shift of the vector representation of that element. Consequently, the inverse of an element $\overline{x} \in \mathbb{F}_2^{17}$, can be computed by means of some shifts (negligable in hardware) together with 4 multiplications in $\mathbb{F}_2^{17}$.

$$\begin{aligned}
\overline{x}^{-1} &= (\overline{x}^{2^1} \cdot \overline{x}^{2^2}) \cdot (\overline{x}^{2^3} \cdot \overline{x}^{2^4}) \cdot (\overline{x}^{2^5} \cdot \overline{x}^{2^6}) \cdot (\overline{x}^{2^7} \cdot \overline{x}^{2^8}) \\
&\quad \cdot (\overline{x}^{2^9} \cdot \overline{x}^{2^{10}}) \cdot (\overline{x}^{2^{11}} \cdot \overline{x}^{2^{12}}) \cdot (\overline{x}^{2^{13}} \cdot \overline{x}^{2^{14}}) \cdot (\overline{x}^{2^{15}} \cdot \overline{x}^{2^{16}}) \\
&= (\overline{y} \cdot \overline{y}^{2^2}) \cdot (\overline{y}^{2^4} \cdot \overline{y}^{2^6}) \cdot (\overline{y}^{2^8} \cdot \overline{y}^{2^{10}}) \cdot (\overline{y}^{2^{12}} \cdot \overline{y}^{2^{14}}) \text{ with } \overline{y} = \overline{x}^{2^1} \cdot \overline{x}^{2^2} \qquad (19) \\
&= (\overline{z} \cdot \overline{z}^{2^4}) \cdot (\overline{z}^{2^8} \cdot \overline{z}^{2^{12}}) \text{ with } \overline{z} = \overline{y} \cdot \overline{y}^{2^2} \\
&= (\overline{w} \cdot \overline{w}^{2^8}) \text{ with } \overline{w} = \overline{z} \cdot \overline{z}^{2^4} .
\end{aligned}$$

A similar implementation is possible, by means of 4 multiplications in $\mathbb{F}_2^{16}$, for the power function 511 in 16 variables. The output bit of this S-box has degree 8, AI equal to 7, and $\epsilon = 2^{-8}$. However, the disadvantage of these functions against the inverse function in 16 variables is that we do not know any efficient masking scheme for the above implementation.

– Another interesting class of functions for hardware implementation are the symmetric functions. These functions satisfy the property that the output is completely determined by the weight of the input vector. Therefore, the truth table $v_f = (v_\varphi, \ldots, v_0)$, also called *value vector*, of the symmetric function f on $\mathbb{F}_2^\varphi$ reduces to a vector of length $\varphi + 1$, corresponding with the function values v_i of the vectors of weight i with $0 \leq i \leq \varphi$. It is easy to check that the following class of symmetric functions have maximum AI:

Theorem 2. *Let f be a symmetric Boolean function on $\mathbb{F}_2^\varphi$ for $\varphi \geq 2$, with value vector $v_f(\overline{x}) = 0$ if $\mathrm{wt}(\overline{x}) \leq \lceil \frac{\varphi}{2} \rceil - 1$ and 1 else. Then the AI of f is equal to $\lceil \frac{\varphi}{2} \rceil$.*

By Proposition 2 and Proposition 4 of [7], the degree of these functions are determined as follows.

Theorem 3. *For $\varphi = 2^i - 1$ with $i \in \mathbb{N}$, the function defined in Theorem 2 is equal to the homogeneous symmetric polynomial of degree 2^{i-1}. For all other dimensions φ with $2^i \leq \varphi < 2^{i+1} - 1$, the degree is equal to 2^i. Moreover, the ANF of the functions in dimensions $\varphi = 2i + 1$ and $\varphi = 2i + 2$, where $2i$ is no power of 2, coincides.*

If φ is odd, these functions are trivially balanced because $v_f(i) = v_f(\varphi - i)$ for $0 \leq i \leq \lfloor \frac{\varphi}{2} \rfloor$. For φ even, the functions are not balanced. But by XORing with an extra input variable from the LFSR, this property is immediately obtained. However, the nonlinearity of these functions is not high. In particular, it holds that $\max_{\overline{w} \in \mathbb{F}_2^\varphi} |W_f(\overline{w})| = 2\binom{\varphi-1}{\frac{\varphi-1}{2}}$ for odd φ and equal to $\binom{\varphi}{\frac{\varphi}{2}}$ for even φ. Therefore, $\epsilon \approx 2^{-3.15}, 2^{-3.26}, 2^{-3.348}$ for $\varphi = 13, 15, 17$ respectively and $\epsilon \approx 2^{-3.15}, 2^{-3.26}$, $2^{-3.348}$ for $\varphi = 12, 14, 16$ respectively.

We suggest to exploit the high AI of these functions together with the high nonlinearity of the inverse function by taking the direct sum of both. Let us recall the following properties of the direct sum of two functions.

Lemma 1. *Assume f is equal to $f(x, y) = f_1(x) \oplus f_2(y) : \mathbb{F}_2^{\varphi_1 + \varphi_2} \to \mathbb{F}_2$. Then the AI of f satisfies $\max\{AI(f_1), AI(f_2)\} \leq AI(f) \leq AI(f_1) + AI(f_2)$. For the Walsh spectrum, it holds that $W_f(w_1, w_2) = W_{f_1}(w_1) W_{f_2}(w_2)$ and thus $\epsilon_f = \frac{\epsilon_{f_1} \epsilon_{f_2}}{2}$.*

If f_1 is the output function of the inverse S-box and f_2 the symmetric function as defined above, both in 16 variables, then the function $f_1(\overline{x}) \oplus f_2(\overline{y})$, where $\overline{x}$ and $\overline{y}$ are two different sets of inputs, have AI greater or equal than 8 and $\epsilon \approx 2^{-10.348}$. This function has the additional implementation cost of the symmetric function but can still be efficiently masked due to the property that $\mathrm{wt}(\overline{x} \oplus \overline{a}) = \mathrm{wt}(\overline{x}) + \mathrm{wt}(\overline{a}) - 2\,\mathrm{wt}(\overline{x} \cdot \overline{a})$.

9 Conclusion

The aim of this paper was to explore the possibility for constructing a basic LFSR-based stream cipher, which is secure, has a low hardware cost and a reasonable throughput.

We have opted for a regularly clocked LFSR-based filter generator. Such a design is very transparent, and is one of the few designs for which a very complete analysis of the linear biases, correlations and nonlinear relations is possible.

We have brought back the security of this scheme with respect to various crypt-analytic attacks to the properties of the LFSR and the Boolean function used. Then, building blocks that achieve a good tradeoff between security and performance in hardware have been chosen. We also described an authentication mechanism that is closely associated with the stream cipher. We believe that the result, SFINKS, can be useful in a number of restricted hardware environments where encryption and authentication are required.

We believe further investigation of LFSR-based schemes is an interesting research topic. Especially the framework for the recent fast algebraic attacks needs to be completed.

An important observation we made is that many of the attacks proposed require an important amount of precomputation. It would be interesting if within ECRYPT a discussion would be done on the relevance of such attacks. The same holds for attacks with long key stream requirements.

References

1. Mehdi-Laurent Akkar and Christophe Giraud. An implementation of DES and AES, secure against some attacks. In C. Koc, D. Naccache, and C. Paar, editors, *Proceedings of CHES'01*, number 2162 in Lecture Notes in Computer Science, pages 309–318. Springer-Verlag, 2001.
2. Frederik Armknecht, Joseph Lano, and Bart Preneel. Extending the resynchronization attack. In Helena Handschuh and Anwar Hasan, editors, *Selected Areas in Cryptography, SAC 2004*, number 3357 in Lecture Notes in Computer Science, pages 19–38. Springer-Verlag, 2004.
3. Steve Babbage. Space/time trade-off in exhaustive search attacks on stream ciphers. Eurocrypt Rump session, 1996.
4. Lejla Batina, Joseph Lano, Nele Mentens, Siddika Berna Ors, Bart Preneel, and Ingrid Verbauwhede. Energy, performance, area versus security trade-offs for stream ciphers. In *State of the Art of Stream Ciphers*, pages 302–310, 2004.
5. Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In B. Schneier, editor, *Fast Software Encryption, FSE 2000*, number 1978 in Lecture Notes in Computer Science, pages 1–18. Springer-Verlag, 2000.
6. Anne Canteaut and Michael Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000*, number 1807 in Lecture Notes in Computer Science, pages 573–588. Springer-Verlag, 2000.
7. Anne Canteaut and Marion Videau. Symmetric Boolean functions. *IEEE Trans. Inform. Theory*, 2004. Regular paper. To appear.
8. Nicolas Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, number 2729 in Lecture Notes in Computer Science, pages 176–194. Springer-Verlag, 2003.
9. Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In E. Biham, editor, *Advances in Cryptology - EUROCRYPT 2003*, number 2656 in Lecture Notes in Computer Science, pages 345–359. Springer-Verlag, 2003. extended version on eprint.
10. Joan Daemen, Rene Govaerts, and Joos Vandewalle. Resynchronization weaknesses in synchronous stream ciphers. In T. Helleseth, editor, *Advances in Cryptology - EUROCRYPT 1993*, number 765 in Lecture Notes in Computer Science, pages 159–167. Springer-Verlag, 1993.

11. Christophe De Cannière, Joseph Lano, and Bart Preneel. Comments on the rediscovery of time memory data tradeoffs. ECRYPT document, 2005. http://ecrypt.eu.org/stream.
12. Hakan Englund and Thomas Johansson. A new simple technique to attack filter generators and related ciphers. In Helena Handschuh and Anwar Hasan, editors, *Selected Areas in Cryptography, SAC 2004*, number 3357 in LNCS, pages 39–53. Springer, 2004.
13. Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *International Symposium on Symbolic and Algebraic Computation — ISSAC 2002*, pages 75–83. ACM Press, 2002.
14. Eric Filiol. Decimation attack of stream ciphers. In E. Okamoto B. Roy, editor, *Progress in Cryptology - INDOCRYPT 2000*, number 1977 in Lecture Notes in Computer Science, pages 31–42. Springer-Verlag, 2000.
15. Joanne Fuller and William Millan. Linear redundancy in S-boxes. In Thomas Johansson, editor, *Fast Software Encryption, FSE 2003*, number 2887 in Lecture Notes in Computer Science, pages 74–86. Springer-Verlag, 2003.
16. Jovan Golic. Computation of low-weight parity-check polynomials. *Electronics Letters*, 32(21):1981–1982, 1996.
17. Jovan Golic. On the security of nonlinear filter generators. In D. Gollman, editor, *Fast Software Encryption, FSE 1996*, number 1039 in Lecture Notes in Computer Science, pages 173–188. Springer-Verlag, 1996.
18. Jovan Golic. Cryptanalysis of alleged A5 stream cipher. In W. Fumy, editor, *Advances in Cryptology - EUROCRYPT 1997*, number 1233 in Lecture Notes in Computer Science, pages 239–255. Springer-Verlag, 1997.
19. Solomon Golomb. *Shift Register Sequences*. Aegean Park Press, 1982.
20. Kishan Gupta and Subhamoy Maitra. Multiples of primitive polynomials over $GF(2)$. In C. Rangan and C. Ding, editors, *Progress in Cryptology - INDOCRYPT 2001*, number 2247 in Lecture Notes in Computer Science, pages 62–72. Springer-Verlag, 2001.
21. Jin Hong and Palash Sarkar. Rediscovery of time memory tradeoffs. Cryptology ePrint Archive, Report 2005/090, 2005. http://eprint.iacr.org/.
22. Antoine Joux and Frederic Muller. A chosen iv attack against turing. In Mitsuru Matsui and Robert Zuccherato, editors, *Selected Areas in Cryptography, SAC 2003*, number 3006 in LNCS, pages 194–207. Springer, 2003.
23. Edwin Key. An analysis of the structure and complexity of nonlinear binary sequence generators. *IEEE Transactions on Information Theory*, 22:732–736, 1976.
24. Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology - CRYPTO 1999*, number 1666 in Lecture Notes in Computer Science, pages 388–397. Springer-Verlag, 1999.
25. Matthias Krause. BDD-based cryptanalysis of keystream generators. In Lars Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002*, number 2332 in Lecture Notes in Computer Science, pages 222–237. Springer-Verlag, 2002.
26. Hugo Krawczyk. Lfsr-based hashing and authentication. In Y. Desmedt, editor, *Advances in Cryptology - CRYPTO 1994*, number 839 in Lecture Notes in Computer Science, pages 129–139. Springer-Verlag, 1994.
27. Joseph Lano, Nele Mentens, Bart Preneel, and Ingrid Verbauwhede. Power analysis of synchronous stream ciphers with resynchronization mechanism. In *State of the Art of Stream Ciphers*, pages 327–333, 2004.
28. Yi Lu and Serge Vaudenay. Cryptanalysis of bluetooth keystream generator two-level E0. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004*, number 3329 in Lecture Notes in Computer Science, pages 483–499. Springer-Verlag, 2004.
29. Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic attacks and decomposition of boolean functions. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, number 3027 in Lecture Notes in Computer Science, pages 474–491. Springer-Verlag, 2004.
30. Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):67–86, 1992.
31. Havard Molland and Thor Helleseth. An improved correlation attack against irregular clocked and filtered keystream generators. In Matthew Franklin, editor, *Advances in Cryptology - CRYPTO 2004*, number 3152 in Lecture Notes in Computer Science, pages 373–389. Springer-Verlag, 2004.
32. National Institute of Standards and Technology. FIPS-46: Data Encryption Standard, January 1977. http://csrc.nist.gov/publications/fips.

33. National Institute of Standards and Technology, National Bureau of Standards, U.S. Department of Commerce. Advanced encryption standard. Federal Information Processing Standard (FIPS) 197, November 2001. http://csrc.nist.gov/publications/fips.
34. Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A side-channel analysis resistant description of the aes s-box. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption, FSE 2005*, Lecture Notes in Computer Science. Springer-Verlag, 2005.
35. Christopher Paar. *Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields*. Doctoral dissertation, Institute for Experimental Mathematics, University of Essen, Germany, 1994.
36. Greg Rose and Philip Hawkes. On the applicability of distinguishing attacks against stream ciphers. In *Proceedings of the 3rd NESSIE Workshop*, page 6, 2002.
37. Rainer Rueppel. Stream ciphers. In G. Simmons, editor, *Contemporary Cryptology. The Science of Information Integrity*, pages 65–134. IEEE Press, 1991.
38. Thomas Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, IT-30(5):776–780, 1984.
39. Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.
40. Yulian Zheng and Xian-Mo Zhang. On plateaued functions. *IEEE Transactions on Information Theory*, 47(3):1215–1223, 2001.

A Hardware Implementation of the Inverse S-box in $\mathbb{F}_{2^{16}}$

Viewing $\mathbb{F}_{(2^m)^n}$ as a field extension of degree n over $\mathbb{F}_{2^m}$, the multiplicative inverse of $\Delta \in \mathbb{F}_{(2^m)^n}$ can be computed as

$$\Delta^{-1} = (\Delta^r)^{-1} \cdot \Delta^{r-1} \text{ with } r = \frac{2^{nm}-1}{2^m-1} \, . \tag{20}$$

This equation consists of operations which can be performed in the subfield $\mathbb{F}_{2^m}$ [35] and can be used recursively to find the inverse in $\mathbb{F}_{(((2^2)^2)^2)^2}$. We use the following notation:

- $\mathbb{F}_{(((2^2)^2)^2)^2}$ is a field extension of degree 2 over $\mathbb{F}_{((2^2)^2)^2}$ constructed using the irreducible polynomial $P(x) = x^2 + p_1 x + p_0$, where x is a root of the corresponding polynomial and $p_1, p_0 \in \mathbb{F}_{((2^2)^2)^2}$.
- $\mathbb{F}_{((2^2)^2)^2}$ is a field extension of degree 2 over $\mathbb{F}_{(2^2)^2}$ using the irreducible polynomial $Q(y) = y^2 + q_1 y + q_0$, with y a root of the polynomial and $q_1, q_0 \in \mathbb{F}_{(2^2)^2}$.
- $\mathbb{F}_{(2^2)^2}$ is a field extension of degree 2 over $\mathbb{F}_{2^2}$ using the irreducible polynomial $R(z) = z^2 + r_1 z + r_0$, with root z and $r_1, r_0 \in \mathbb{F}_{2^2}$.
- $\mathbb{F}_{2^2}$ is a field extension of degree 2 over $\mathbb{F}_2$ using the irreducible polynomial $S(u) = u^2 + u + 1$ with root u.

For our hardware implementation we made the following parameter choices:

- $r_1 = 0, r_0 = u = \lambda_1$;
- $q_1 = 0, q_0 = (u+1)z = \lambda_2$;
- $p_1 = 0, p_0 = uzy = \lambda_3$.

Eq. (20) is implemented as

$$\begin{aligned}
&\Delta_3 = \delta_{31}x + \delta_{30} \in \mathbb{F}_{\left(\left(\left(2^2\right)^2\right)^2\right)^2} : \\
&\quad \Delta_3^{-1} = (\delta_{31}x + (\delta_{31} + \delta_{30})) \cdot (\underbrace{\lambda_3\delta_{31}^2 + (\delta_{31} + \delta_{30})\delta_{30}}_{\Delta_2})^{-1} . \\
&\Delta_2 = \delta_{21}y + \delta_{20} \in \mathbb{F}_{\left(\left(2^2\right)^2\right)^2} : \\
&\quad \Delta_2^{-1} = (\delta_{21}y + (\delta_{21} + \delta_{20})) \cdot (\underbrace{\lambda_2\delta_{21}^2 + (\delta_{21} + \delta_{20})\delta_{20}}_{\Delta_1})^{-1} . \\
&\Delta_1 = \delta_{11}z + \delta_{10} \in \mathbb{F}_{\left(2^2\right)^2} : \\
&\quad \Delta_1^{-1} = (\delta_{11}z + (\delta_{11} + \delta_{10})) \cdot (\underbrace{\lambda_1\delta_{11}^2 + (\delta_{11} + \delta_{10})\delta_{10}}_{\Delta_0})^{-1} .
\end{aligned} \tag{21}$$

An inversion in $\mathbb{F}_{2^2}$ requires only one addition:

$$\Delta_0 = \delta_{01}u + \delta_{00} \in \mathbb{F}_{2^2} : \Delta_0^{-1} = \delta_{01}u + (\delta_{01} + \delta_{00}) . \tag{22}$$

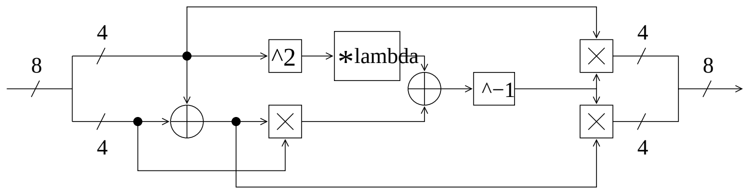

Fig. 2. Architecture of the inversion

The inversion in $\mathbb{F}_{2^{16}}$ is eventually decomposed into operations in $\mathbb{F}_{2^2}$. Therefore a transformation is needed to go from a representation in $\mathbb{F}_{2^{16}}$ (using the irreducible polynomial $t^{16} + t^5 + t^3 + t^2 + 1$) to a representation in $\mathbb{F}_{\left(\left(\left(2^2\right)^2\right)^2\right)^2}$. In [35], Paar explains how a matrix can be created to perform this transformation. Different choices for the irreducible polynomials $P(x)$, $Q(y)$ and $R(z)$ lead to different transformation matrices. For every combination of $P(x)$, $Q(y)$ and $R(z)$ there are 16 possibilities for the transformation matrix. For hardware implementations, the most area efficient transformation matrix is the one that has the least '1' entries, because this number determines the XOR gate count for the transformation. After performing the inversion using the $\mathbb{F}_{\left(\left(\left(2^2\right)^2\right)^2\right)^2}$ representation we need to go back to the $\mathbb{F}_{2^{16}}$ representation using the inverse of the transformation matrix. Our choice for the transformation matrices T and T^{-1} is as follows:

$$T = \begin{bmatrix}
1\,1\,0\,0\,1\,0\,1\,1\,0\,0\,1\,0\,0\,1\,0\,1 \\
0\,1\,1\,1\,1\,0\,1\,1\,0\,0\,0\,1\,0\,0\,0\,0 \\
0\,1\,1\,1\,1\,1\,1\,1\,0\,1\,0\,0\,1\,1\,1\,1 \\
0\,0\,0\,0\,1\,0\,0\,0\,1\,1\,0\,0\,0\,1\,1\,1 \\
0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,1\,1\,0\,0\,1 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,1 \\
0\,0\,1\,0\,0\,1\,0\,1\,0\,1\,1\,0\,0\,0\,1\,0 \\
0\,1\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1 \\
0\,1\,1\,0\,1\,1\,0\,0\,0\,0\,1\,0\,1\,0\,0\,0 \\
0\,1\,0\,1\,1\,1\,0\,1\,0\,1\,0\,1\,1\,0\,0\,0 \\
0\,1\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,1\,1 \\
0\,1\,0\,1\,0\,0\,0\,1\,0\,0\,1\,0\,1\,0\,1\,0 \\
0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,1\,0\,1\,0 \\
0\,0\,1\,0\,0\,1\,0\,1\,1\,0\,0\,0\,0\,1\,0\,0 \\
0\,0\,1\,0\,0\,0\,1\,0\,0\,1\,1\,0\,1\,1\,0\,1 \\
0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,1\,0\,1\,1
\end{bmatrix}, \quad T^{-1} = \begin{bmatrix}
0\,1\,1\,1\,1\,1\,1\,1\,0\,1\,0\,0\,1\,1\,1\,1 \\
0\,0\,0\,0\,1\,0\,0\,0\,1\,1\,0\,0\,0\,1\,1\,1 \\
0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,1\,1\,0\,0\,1 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,1 \\
0\,0\,1\,0\,0\,1\,0\,1\,0\,1\,1\,0\,0\,0\,1\,0 \\
0\,1\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1 \\
0\,1\,1\,0\,1\,1\,0\,0\,0\,0\,1\,0\,1\,0\,0\,0 \\
0\,1\,0\,1\,1\,1\,0\,1\,0\,1\,0\,1\,1\,0\,0\,0 \\
0\,1\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,1\,1 \\
0\,1\,0\,1\,0\,0\,0\,1\,0\,0\,1\,0\,1\,0\,1\,0 \\
0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,1\,0\,1\,0 \\
0\,0\,1\,0\,0\,1\,0\,1\,1\,0\,0\,0\,0\,1\,0\,0 \\
0\,0\,1\,0\,0\,0\,1\,0\,0\,1\,1\,0\,1\,1\,0\,1 \\
0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,1\,0\,1\,1
\end{bmatrix}.$$