Key Recovery Attacks on Grain-like Keystream Generators with Key Injection

Matthew Beighton$^{1[0000-0002-7772-757X]}$, Harry Bartlett*$^{1[0000-0003-4347-0144]}$, Leonie Simpson$^{1[0000-0001-8434-9741]}$, and Kenneth Koon-Ho Wong$^{1[0000-0003-1732-6149]}$

Queensland University of Technology: Brisbane, Queensland, AU
matthew.beighton@hdr.qut.edu.au, *h.bartlett@qut.edu.au, lr.simpson@qut.edu.au, k45.wong@connect.qut.edu.au

Abstract. A common structure in stream ciphers makes use of linear and nonlinear shift registers with a nonlinear output function drawing from both registers. We refer to these as Grain-like keystream generators. A recent development in lightweight ciphers is a modification of this structure to include a non-volatile key register, which allows key bits to be fed into the state update of the nonlinear register. Sprout and Plantlet are examples of this modified structure. The authors of these ciphers argue that including these key bits in the internal state update provides increased security, enabling the use of reduced register sizes below the commonly accepted rule of thumb that the state size should be at least twice the key size.

In this paper, we analyse Plantlet and show that the security of this design depends entirely on the choice of the output function. Specifically, the contribution from the nonlinear register to the output function determines whether a key recovery attack is possible. We make a minor modification to Plantlet's output function which allows the contents of the linear register to be recovered using an algebraic attack during keystream generation. This information then allows partial recovery of the contents of the nonlinear register, after which the key bits and the remaining register contents can be obtained using a guess and check approach, with a complexity significantly lower than exhaustive key search.

Note that our attack is not successful on the existing version of Plantlet, though it only requires minor modifications to the filter function in order for the attack to succeed. However, our results clearly demonstrate that including the key in the state update during keystream generation does not increase the security of Plantlet. In fact, this feature was exploited to recover the key during keystream generation without the need to consider the initialisation process. This paper provides design guidelines for choosing both suitable output functions and the register stages used for inputs to these functions in order to resist the attacks we applied.

Keywords: Key recovery · algebraic attack · key injection · Plantlet · Grain-like structures · lightweight ciphers

1 Introduction

Symmetric stream ciphers are used to provide confidentiality for a range of real-time applications. The most common type of stream cipher is the binary additive stream cipher, where encryption and decryption are performed by XORing a binary keystream with the plaintext or ciphertext bitstream, respectively. The reciprocal nature of the XOR operation provides high speed encryption and decryption processes. However, the security provided depends on the properties of the keystream. The security of the keystream generator is therefore crucial.

The importance of secure encryption is highlighted by the recent NIST competition for AEAD algorithms suitable for lightweight applications. These algorithms are intended for use in applications such as the ubiquitous Internet of Things (IoT) and require high security levels with reduced computational load and/or component size.

Many keystream generators are based on shift registers with either a linear or nonlinear feedback function, denoted as linear feedback shift registers (LFSRs) and nonlinear feedback shift registers (NFSRs), respectively. However, previous work [3, 6, 7] has shown that shift register based ciphers are vulnerable to algebraic attacks. In response, some contemporary keystream generators use a combination of a NFSR and a LFSR, together with a nonlinear filter function taking inputs from both registers. We refer to these generators as "Grain-like structures", as the well known Grain family of stream ciphers [13–17] is designed in this way. The combination of LFSRs and NFSRs in these structures was intended to prevent such algebraic attacks; however, Berbain et al [4] and Beighton et al [2] have demonstrated viable attacks against Grain-like structures with particular forms of output function.

A further development, aimed at providing encryption for lightweight devices, is to include a non-volatile key register in the cipher state. The key bits stored in this register are then injected into the feedback of the shift registers during operation. The ciphers Sprout [1] and Plantlet [22] both use this technique. This introduction of key injection into Grain-like ciphers was intended to increase the security of these ciphers, so that lightweight ciphers with smaller registers could still provide acceptable security levels. In particular, the internal state size in these ciphers is less than twice the key size, contradictory to the rule of thumb from Hong and Sarkar [18] in relation to generic time-memory trade-off attacks.

The authors of Sprout [1] invited further investigation of these designs featuring key injection. In this paper, we therefore investigate the security provided by Grain-like keystream generators with key injection. Specifically, we explore an algebraic key recovery attack on Plantlet and find that it can be successfully performed with only a minor modification to the output function and with the key injection feature left intact. Our attack is based on the algebraic attack applied to Grain-like structures by Beighton et al [2]; it is applied during keystream generation, so no knowledge of the initialisation function is required.

This paper is organised as follows: Section 2 provides background information on shift register based designs. Section 3 discusses current algebraic attack techniques. Section 4 presents our algebraic attack technique for application to Grain-like structures with key injection. We then apply our attack technique to modified Plantlet in Section 5. Experimental simulations for proof of concept are reported in Section 6 and discussed in Section 7. Conclusions are drawn in Section 8.

2 Preliminaries and Notation

2.1 Feedback Shift Registers

A binary feedback shift register (FSR) of length n is a set of n storage devices called *stages* $(r_0, r_1, ..., r_{n-1})$, each containing one bit, together with a Boolean update function g. The state at any time t is defined to be S_t, where $S_t = s_t, s_{t+1}, ..., s_{t+(n-1)}$, and the sequence of state bits that passes through the register over time is denoted S; that is $S = s_0, s_1, .., s_{n-1}, s_n, \dots$.

The shift registers used in the types of keystream generators we discuss in this paper are regularly clocked Fibonacci style, as shown in Figure 1. Thus, the state update function takes the form:

$$r_i^{t+1} = \begin{cases} r_{i+1}^t & i = 0, 1, \dots, n-2 \\ g(r_0, r_1, .., r_{n-1}) & i = n-1 \end{cases}$$

If g is linear, the register is said to be a linear feedback shift register (LFSR) and if g is nonlinear, the register is said to be a nonlinear feedback shift register (NFSR).

A binary sequence can be generated from a FSR by applying a Boolean function f to the state S_t, as shown in Figure 1. Here, the output $y = f(S_t)$ can be a function of the contents of one or more register stages.

2.2 Filter Generators

Keystream generators where f is a function of the contents of multiple stages are called *filter generators*. If f and g are both linear, the filter

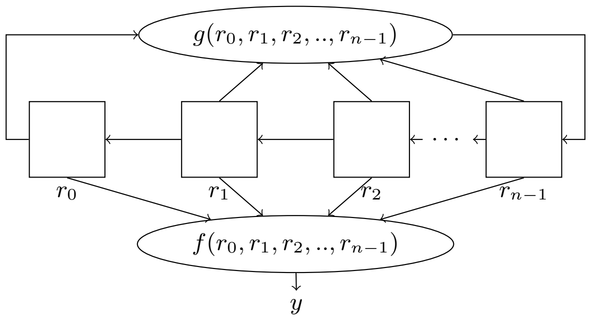
$g(r_0, r_1, r_2, .., r_{n-1})$
r_0
r_1
r_2
r_{n-1}
$f(r_0, r_1, r_2, .., r_{n-1})$
y

Fig. 1. An n-stage FSR with update function g and filter function f.

generator is equivalent to another LFSR, which provides very little security to the plaintext [20]. For this reason, LFSRs were traditionally filtered using a nonlinear Boolean function [19].

A keystream generator consisting of a LFSR and a nonlinear filter function f is known as a *nonlinear filter generator* (NLFG) [24]. These designs have been extensively analysed and are susceptible to numerous attacks, including correlation attacks [24, 10, 21, 12], algebraic attacks [7, 9, 6] and distinguishing attacks [8]. The underlying LFSR provides only desirable statistical properties for the binary sequence, while the resistance of the NLFG to cryptanalysis is determined by the properties of the nonlinear filter function. As a single nonlinear Boolean function cannot display high levels of all the desirable cryptographic properties [23], choosing a filter function that resists one form of attack may leave the keystream generator vulnerable to other attacks.

In response to the cryptanalysis of NLFGs, designs using NFSRs were proposed. A *linearly filtered nonlinear feedback shift register* (LF-NFSR) [11] has a nonlinear update function g and a linear filter function f, and is the dual construction of the NLFG. Berbain et al. [3] showed that LF-NFSRs are also susceptible to algebraic attacks, resulting in initial state (and possibly secret key) recovery. From Berbain's results, it is clear that the properties of the filter function used in a LF-NFSR are critical in providing resistance to a traditional algebraic attack.

2.3 Composite combiners and 'Grain-like' structures

Effective algebraic attacks have been proposed on both NLFG and LF-NFSR keystream generators. A more complex design incorporates both a LFSR and a NFSR, together with a nonlinear filter function taking inputs from both registers, as shown in Figure 2. Keystream generators using this structure include Grain [15] and subsequent variants of Grain [13, 14, 16, 17]. We denote this general design as a "Grain-like" structure. Here we consider the lengths of the NFSR and LFSR to be the same (n). However, the approach outlined also applies in the case where register lengths differ.

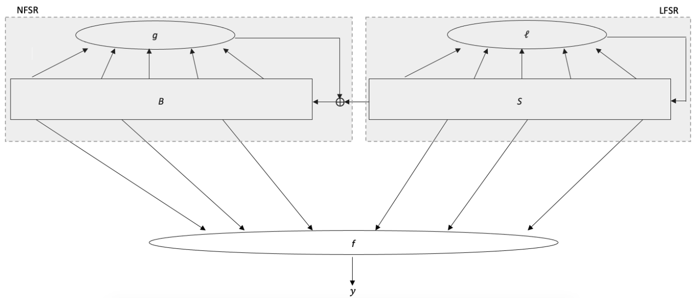

Fig. 2. Grain-like structure.

We denote the states of the NFSR and the LFSR at any time t as B_t and S_t. The sequences of state bits that pass through the registers over time are denoted B and S; that is $B = b_0, b_1, .., b_{n-1}, b_n, ...$ and $S = s_0, s_1, .., s_{n-1}, s_n, ...$. In the case of Grain-like structures, we denote the nonlinear update function as g, the linear update function as ℓ and the filter function as f. For a Grain-like structure, the LFSR is autonomous when producing output as all of the inputs to ℓ are from the LFSR. The NFSR is not autonomous, as the nonlinear update function g contains one input from the LFSR.

The filter function f can be considered as the XOR sum (here denoted '+') of several different types of monomials. That is, we consider sub-functions of f. We define the following sub-functions, each as a sum of the terms indicated:

- L_B - monomials with linear inputs from NFSR.
- L_S - monomials with linear inputs from LFSR.
- f_S - nonlinear monomials with inputs from LFSR only.
- f_B - nonlinear monomials with inputs from NFSR only.
- f_{BS} - nonlinear monomials with inputs from both NFSR and LFSR.

Thus, any filter function f in a Grain-like structure can be expressed as follows:

$$f(B, S) = L_B + L_S + f_B + f_S + f_{BS}. \quad (1)$$

2.4 Grain-like structures using key injection

Newer contemporary designs have been proposed which use the general Grain-like structure, but also incorporate the secret key in the state update of the keystream generator, as in Figure 3. Keystream generators such as those used in Sprout [1] and Plantlet [22] have this design. We denote this general design as a "Grain-like structure with key injection" and use the same notation as for a general Grain-like structure.

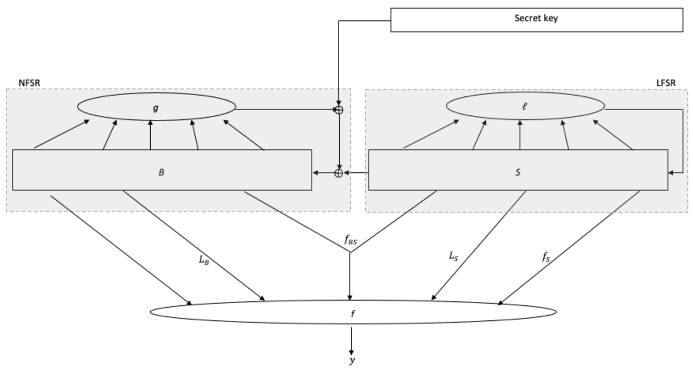

Fig. 3. Grain-like structure with key injection.

3 Current Algebraic Attacks

Algebraic attacks were first introduced by Courtois and Meier [7] on ciphers with linear feedback. The goal of an algebraic attack is to create a system of low degree equations that relates the initial state bits of the cipher to some observed output bits and then to solve these equations to recover the internal state values. For a binary additive stream cipher, the output may be obtained using a known-plaintext attack.

These attacks are performed in two phases: pre-computation and online. The pre-computation phase builds a system of equations relating initial state bits and output bits. In the online phase, given an observed output sequence $\{y_t\}_{t=0}^{\infty}$, the appropriate substitutions are performed, the system is solved and the initial state recovered.

3.1 Algebraic attacks on NLFGs [7]

Each output bit of a NFSR satisfies the equation

$$y_t = f(S_t) = f(s_t, \ldots, s_{t+n-1}) \tag{2}$$

The linear update function g of the LFSR can then be used to replace state bits s_{t+n-1} with the corresponding linear combination of initial state bits, keeping the equation system of a constant degree ($deg(f)$) while maintaining the number of unknown variables in the system.

In many cases, each equation in the system may be multiplied through by a low degree multivariate function h (of degree e) to reduce the overall degree of the system of equations [7]. If $fh = 0$, then h is defined as an annihilator of f. Each equation in the resulting system has the form

$$f(S_t)h(S_t) = y_t h(S_t).$$

The degree of this system will be equal to $deg(fh) = d$, where $d < deg(f)$, with n independent variables, where n is the length of the underlying LFSR. For a more detailed explanation, the reader is referred to Courtois and Meier's paper [7]. Appendix A.1 provides an algorithm for this attack and also discusses data requirements and computational complexity.

3.2 Fast algebraic attacks on NLFGs [6]

Fast algebraic attacks [6] significantly reduce the complexity of the online phase by reducing the overall degree of the system of equations below the degree of fh. This increases the complexity of the precomputation phase; however, precomputation only needs to be performed once for a particular cipher and the equation system may then be reused in multiple online phases to recover the states of the cipher corresponding to multiple different keystreams.

These attacks use a concept Courtois [6] described as "double-decker equations". These equations allow an attacker to equate an expression in terms of initial state bits only to an expression in terms of initial state bits and observed output bits. The technique targets monomials in initial state bits only of degree from $e = deg(h)$ to $d = deg(fh)$: given approximately $\binom{n}{d}$ equations, these monomials will occur in multiple equations and can be replaced by suitable linear combinations of those equations. These linear combinations define a new system in n unknowns, of degree $e < d$. This new system can be solved by linearisation, with less computational complexity than for traditional algebraic attacks.

For a detailed explanation, the reader is referred to Courtois' paper [6]. Appendix A.2 provides an algorithm for this attack and also discusses data requirements and computational complexity.

3.3 Algebraic attacks on LF-NFSRs

Initially, LF-NFSRs were considered resistant to algebraic attacks, due to the use of a nonlinear state update function. Using the nonlinear feedback function to derive equations for the update bits in terms of initial state bits causes the degree of the system of equations to increase over time. However, Berbain et al. [3] showed that it is possible to keep the degree of the system of equations constant. This allows an algebraic attack which can recover the initial state of the underlying NFSR (and possibly the secret key).

Berbain et al. [3] noted that the equation for the first output bit

$$y_0 = \ell(s_0, ..., s_{n-1}) = \sum_{k=0}^{n-1} a_k s_k,$$

(with $a_k \in \{0, 1\}$) implies that

$$s_j = \sum_{k=0}^{j-1} a_k s_k + y_0.$$

where j is the highest index in the original summation for which $a_j = 1$.

Repeating this process for all subsequent time steps allows us to express every bit, s_{j+t} for $t \geq 0$, as a linear combination of output bits and initial state bits. This produces a set of equations of the form:

$$s_{j+t} = \sum_{k=0}^{j-1} a_{k+t} s_{k+t} + y_t,$$

for $t \geq 0$. Note that if the latter summations contain any term for which an equation already exists, the term can be replaced by the corresponding linear combination of initial state bits and output bits.

Appendix A.3 provides an algorithm for this attack and also discusses data requirements and computational complexity.

3.4 Algebraic attacks on Grain-like structures

After successfully applying algebraic attacks to LF-NFSRs, Berbain et al. [4] proposed an algebraic attack on Grain-like structures where the output function f is the XOR combination of a LF-NFSR and a NLFG. That is, adopting the notation from Section 2.3, $f(B, S) = L_B + L_S + f_S$.

In this case the output of the keystream generator can be expressed as

$$y_0 = L_B + L_S + f_S = \sum_{k=0}^{n-1} a_k b_k + L_S + f_S. \tag{3}$$

As discussed in Section 3.3, an equation of the form taken by Equation 3 can be rearranged as follows:

$$b_j = \sum_{k=0}^{j-1} a_k b_k + L_S + f_S + y_0.$$

Repeating this for $t > 0$ allows for NFSR state bits of index j or higher to be represented as the XOR sum of:

- a linear combination of NFSR initial state bits
- a linear and nonlinear combination of LFSR initial state bits
- a linear combination of observed output bits.

A second system of equations can then be built using the nonlinear update function to the NFSR, making substitutions from the system generated by Equation 3 where applicable. This system will be of degree at most $deg(g)deg(f_S)$. Combining the two systems results in a system of equations of degree $deg(g)deg(f_S)$ in $n + j$ unknown initial state bits, where n is the size of LFSR and j is the index of the highest indexed term in L_B. The success of this attack in recovering the LFSR and NFSR initial states demonstrated that using the contents of stages in the NFSR linearly in the output function is not sufficient to provide resistance to algebraic attacks; the NFSR contents must also be filtered nonlinearly in some way.

Beighton et al. [2] proposed an algebraic attack on Grain-like structures where the output function f is of the form $f = L_S + f_S + f_{BS}$. They note that, by using the idea of annihilators presented by Courtois and Meier [7], f may be multiplied by a low degree function containing only LFSR bits that will eliminate f_{BS}. This function, denoted A_{BS}, is considered as a "partial annihilator", in as much as A_{BS} only annihilates certain monomials.

Multiplying f by A_{BS} leaves an equation of the form

$$zA_{BS} = A_{BS}(L_S + f_S),$$

which is an equation containing only LFSR initial state bits. Fast algebraic attack techniques can then be applied to recover the LFSR initial state, from which the NFSR initial state can be partially recovered.

4 Our divide and conquer attack on Grain-like structures with key injection

As highlighted in Beighton et al's paper [2], if the filter function of a Grain-like structure does not feature a monomial that takes inputs only from the NFSR and does not divide any other monomial, the structure is vulnerable to a divide and conquer attack which first targets the LFSR in an algebraic attack and then determines the NFSR contents.

This idea extends to the case where the secret key is used to update the NFSR, as the LFSR still runs autonomously.

4.1 Generalised algebraic attack algorithm

We present here the generalised algebraic attack for Grain-like structures with key injection. This attack uses a divide and conquer strategy. We first target the LFSR and recover the LFSR initial state. The NFSR is then targeted, with partial NFSR intitial state recovery possible. From this point, we can simultaneously determine the key bits and the remaining NFSR bits.

Recovering the LFSR Consider a keystream generator that produces an output bit at each time step by:

$$z = L_S + f_S + f_{BS}. \tag{4}$$

That is, NFSR state bits are only used nonlinearly and only in f_{BS}. Every monomial in f_{BS} will contain both NFSR bits and LFSR bits. Thus, using the idea of annihilators presented by Courtois and Meier [7], we may multiply Equation 4 by a low degree function containing only LFSR bits that will eliminate f_{BS}. We denote this function as A_{BS}, and consider it to be a "partial annihilator" in as much as A_{BS} only annihilates certain monomials. Note that the degree of the NFSR bits in f_{BS} does not affect the ability to annihilate the monomials containing bits from the NFSR.

Therefore Equation 4 can be rewritten as

$$zA_{BS} = A_{BS}(L_S + f_S), \tag{5}$$

which is an equation containing only LFSR initial state bits. The degree of the system of equations built using Equation 5 will be at most $deg(A_{BS})+ deg(f_S)$. Note, however, that the right hand side of Equation 5 contains only initial state bits from the LFSR. This means that fast algebraic attack methods can be performed in the precomputation phase of the attack to reduce the degree of unknown variables in the system from $deg(A_{BS}) + deg(f_S)$ to $deg(A_{BS})$.

There are several other cases where the attack works. For convenience we use only the simplest example here to illustrate the first phase of the attack and refer the reader to Beighton et al's paper [2] for a detailed discussion of the other cases.

The structure of a system of equations built in this way allows for the fast algebraic attack techniques highlighted in Section 3.2 to be applied. That is, given access to approximately $\binom{n}{d}$ bits of output (where n is the size of the LFSR and d is the algebraic degree of the system relating LFSR initial state bits to observable output bits), a precomputation phase can be performed that allows a new system of equations to be built of degree $e < d$, where e is the degree of A_{BS}. This precomputation phase has a complexity of $\mathcal{O}(\binom{n}{d} log\binom{n}{d} + n\binom{n}{d})$. The initial state of the LFSR can then be recovered in the online phase of the attack by observing approximately $\binom{n}{d}$ bits of output with complexity $\mathcal{O}(\binom{n}{d}\binom{n}{e} + \binom{n}{e}^{\omega})$, where ω is the Guassian elimination exponent $\omega \approx 2.8$.

Recovering the NFSR and the key Once the LFSR initial state is recovered, every future LFSR state bit will be known, as the LFSR is autonomous during keystream generation. The next stage is to recover the NFSR initial state. In doing so, we will simultaneously recover the key.

Since the key is used to update the state of the NFSR, recovering the NFSR state at one point in time without knowing the key is not enough to determine the NFSR state at other points in time. As a result, the key must also be taken into account. To begin, however, we will consider the idea of recovering the NFSR state by itself and then expand this approach to recover both the NFSR state and the key.

Consider the example filter function used to produce the keystream bit

$$z = x_1 + x_4x_5 + x_0x_3 + x_0x_1x_2 + x_2x_3x_4x_5, \tag{6}$$

where x_0, x_1, x_2, x_3 are from the LFSR and x_4, x_5 are from the NFSR. Since the LFSR is known, each output bit will have the form

$$z = \alpha x_4x_5 + \beta,$$

where α and β may be 0 or 1, respectively.

Clearly, when $\alpha = 0$ no information about the initial state of the NFSR is leaked. We must therefore utilise the case where $\alpha = 1$. If $z =$

x_4x_5 and $z = 1$, then we know $x_4 = x_5 = 1$. Likewise if $z = x_4x_5 + 1$ and $z = 0$, then we know $x_4 = x_5 = 1$. Once we have recovered these state bits, we may then look to equations where $z = x_4x_5$ and $z = 0$, but for which we know either x_4 or x_5 equals 1. We would then know that the unknown state bit is equal to zero. Similarly for the case where $z = x_4x_5 + 1$ and $z = 1$. Continuing in this way, we may be able to recover n consecutive bits of the NFSR.

For certain filter functions it may not be possible to recover n consecutive state bits. In this case, the partially recovered initial state reduces the exhaustive search required to recover the correct initial state of the NFSR. For instance, suppose m bits of the NFSR can be recovered. This leaves 2^{n-m} possible candidates for the correct NFSR initial state which, for $m > 0$, is better than exhaustively searching the entire register. Each candidate can be used (together with the known LFSR initial state) to produce output. The candidate which produces the correct output sequence can be assumed to be the correct initial state.

As stated earlier, because the key is used to update the state of the NFSR, recovering the state of the NFSR at a single point in time is not sufficient to determine the state at any other point in time. However, as we now show, the NFSR state recovery process discussed above may be adapted to recover the NFSR state and the secret key simultaneously by considering a longer sequence of NFSR bits. We consider the sequence of bits that pass through the NFSR as the XOR of some NFSR feedback bits and the key. We thus have,

$$\begin{aligned} b_{t+n} &= g(B_t) + k_{t \bmod |K|} \\ &= b'_{t+n} + k_{t \bmod |K|}. \end{aligned} \tag{7}$$

Thus, given $n + |K|$ consecutive NFSR state bits, the key can easily be recovered.

The goal is therefore to recover $n + |K|$ consecutive bits of the NFSR, or to recover as much as is possible and then exhaustively search the rest. For instance, suppose m bits of the NFSR sequence can be recovered. This leaves $2^{(n+|K|)-m}$ possible candidates for the correct NFSR sequence which, for $m > n$, is better than exhaustively searching the key. Each candidate can be used (together with the known LFSR initial state) to produce further NFSR states and thus output. The candidate which produces the correct output sequence can be assumed to be the correct sequence. From the sequence the key can quickly be recovered.

5 Algebraic Attack on a Modified Version of Plantlet

We now mount an algebraic attack on adapted versions of the stream cipher Plantlet with a modified filter function. We show that even with the key bits used to update the NFSR state, the success of the attack is determined only by the choice of filter function.

5.1 The Plantlet stream cipher

Plantlet is a contemporary stream cipher that uses a very small internal state. The keystream generator for Plantlet consists of a LFSR and a NFSR, together with a nonlinear Boolean function that takes inputs from both registers.

Initialisation Plantlet takes as input a 80-bit secret key and 90-bit IV. In initialisation, the NFSR and the LFSR are 40 bits and 60 bits in length. The keystream generator is loaded by filling the NFSR with 40 IV bits. The remaining 50 bits of the IV are loaded into the LFSR, which is then padded with a constant.

At each time step the LFSR is updated using the linear update function ℓ as follows

$$s_{t+59} = \ell(S_t) = z_t + s_t + s_{t+14} + s_{t+20} + s_{t+34} + s_{t+43} + s_{t+54}. \tag{8}$$

The NFSR is updated at each time step using the nonlinear update function g as follows

$$\begin{aligned} b_{t+39} = g(B_t) = {} & s_t + z_t k_{t \bmod 80} + c_t^4 + b_t + b_{t+13} + b_{t+19} + b_{t+35} + b_{t+39} \\ & + b_{t+2}b_{t+25} + b_{t+3}b_{t+5} + b_{t+7}b_{t+8} + b_{t+14}b_{t+21} + b_{t+16}b_{t+18} \\ & + b_{t+22}b_{t+24} + b_{t+26}b_{t+32} + b_{t+33}b_{t+36}b_{t+37}b_{t+38} \\ & + b_{t+10}b_{t+11}b_{t+12} + b_{t+27}b_{t+30}b_{t+31}, \end{aligned} \tag{9}$$

where c_t^4 is the fourth least-significant-bit of the modulo 80 counter. This counter is public so, for convenience, it is easier to combined the XOR of the counter variable and the key variable into one variable as follows

$$k'_t = k_{t \bmod 80} + c_t^4. \tag{10}$$

Keystream is not produced in initialisation. Instead, the output of the filter function is used to update the state. Output is produced using the following filter function

$$\begin{aligned} z_t = s_{t+30} + b_{t+1} + b_{t+6} + b_{t+15} + b_{t+17} + b_{t+23} + b_{t+28} + b_{t+34} + b_{t+4}s_{t+6} \\ + s_{t+8}s_{t+10} + s_{t+32}s_{t+17} + s_{t+19}s_{t+23} + b_{t+4}s_{t+32}b_{t+38}. \end{aligned} \tag{11}$$

Keystream generation At the end of initialisation, the LFSR is increased by one bit and the new stage is loaded with a one. Thus, in keystream generation Plantlet consists of a 40-bit NFSR and a 61-bit LFSR. This adjustment to the LFSR is made in order to avoid the possibility of the LFSR being initialised to the all zero state.

The update functions used to update the state of the LFSR and the NFSR during keystream generation are identical to those used in initialisation; however, the output function is now used to generate keystream and so it is not used to update the state.

5.2 Modified version of Plantlet

We introduce a modified version of the Plantlet stream cipher, where we replace any independent linear term taken from the NFSR by the corresponding term in the LFSR. That is, the filter function $f(B, S)$ remains the same, except that monomials appearing only in L_B are replaced by monomials in L_S with the same indices. Note that we denote the modified version of Plantlet by appending the suffix $-m$.
Table 1 highlights the differences between the original and modified versions.

Table 1. Modifications to linear combinations in Plantlet.

Original linear combination	Modified linear combination
$b_{t+1} + b_{t+6} + b_{t+15} + b_{t+17} + b_{t+23} + b_{t+28} + b_{t+34}$	$s_{t+1} + s_{t+6} + s_{t+15} + s_{t+17} + s_{t+23} + s_{t+28} + s_{t+34}$

5.3 Stage 1: LFSR recovery

In this section we apply the algorithm from Section 4. The theoretical data and computational complexity requirements to recover the LFSR initial state is summarised in Table 2. In Section 6, we provide experimental results for the modified version of Plantlet.

At time $t = 0$ an output bit in Plantlet-m is produced as follows:

$$z_0 = s_1 + s_6 + s_{15} + s_{17} + s_{23} + s_{28} + s_{34} + b_4 s_6 + s_8 s_{10} + s_{32} s_{17} + s_{19} s_{23} + b_4 s_{32} b_{38}$$

Multiplying this equation by $(s_6 + 1)(s_{32} + 1)$ gives

$$\begin{aligned}(s_6 + 1)(s_{32} + 1)z_0 = {} & s_1 s_6 s_{32} + s_1 s_6 + s_1 s_{32} + s_1 + s_6 s_{15} s_{32} + s_6 s_{15} + s_6 s_{17} s_{32} \\ & + s_6 s_{17} + s_6 s_{23} s_{32} s_{19} + s_6 s_{23} s_{32} + s_6 s_{23} s_{19} + s_6 s_{23} \\ & + s_6 s_{28} s_{32} + s_6 s_{28} + s_6 s_{34} s_{32} + s_6 s_{34} + s_6 s_8 s_{10} s_{32} \\ & + s_6 s_8 s_{10} + s_{15} s_{32} + s_{15} + s_{17} s_{32} + s_{17} + s_{23} s_{32} s_{19} \\ & + s_{23} s_{32} + s_{23} s_{19} + s_{23} + s_{28} s_{32} + s_{28} + s_{34} s_{32} \\ & + s_{34} + s_8 s_{10} s_{32} + s_8 s_{10}\end{aligned} \tag{12}$$

where the right hand side of the equation contains only LFSR initial state bits and is of degree 4. Thus, by observing at least $\binom{61}{4}$ keystream bits, fast algebraic techniques may be applied in the precomputation phase of the attack to reduce the overall degree of the system to the degree of the left hand side (which is of degree 2 in the unknown LFSR initial state bits) [6].

Table 2. Resource requirements for recovering the LFSR of modified Plantlet.

Precomputation phase	
Degree before fast algebraic techniques	4
Complexity	$\mathcal{O}(2^{23})$
Degree after fast algebraic techniques	2
Online phase	
Data	2^{19}
Complexity	$\mathcal{O}(2^{32})$

5.4 Stage 2: NFSR recovery and key recovery

Once the LFSR initial state is recovered, the output function will contain only unknown state bits from the NFSR. As described in Section 4.1, to be able to predict the NFSR state sequence we must know at least $|NFSR| + |K|$ consecutive bits of the NFSR sequence. For Plantlet, at least $40 + 80 = 120$ consecutive bits of NFSR sequence is required to predict the NFSR sequence. Note that if 120 bits of NFSR sequence is known, the key can easily be calculated. Thus, the goal is to recover 120 bits of consecutive NFSR bits.

The data requirement for this stage will utilise the data collected for LFSR state recovery. The computational complexity to partially recover the NFSR sequence is considered to be negligible [4]. The number of NFSR sequence bits recovered over a 120-bit period through application of this method is hard to estimate and will vary depending on the particular key and IV used. However, some guidance based on experimental results is provided in Section 6.2. Due to the low computational complexity of partial NFSR sequence recovery we provide experimental results for this in the next section.

At time $t = 0$ an output bit in Plantlet-m is produced by :

$$\begin{aligned} z_0 = s_{t+1} + s_{t+6} + s_{t+15} + s_{t+17} + s_{t+23} + s_{t+28} + s_{t+34} + b_{t+4}s_{t+6} \\ + s_{t+8}s_{t+10} + s_{t+32}s_{t+17} + s_{t+19}s_{t+23} + b_{t+4}s_{t+32}b_{t+38} \end{aligned}$$

This function is linear in the NFSR bit b_4 and nonlinear in the NFSR bits b_4 and b_{38}. At each time step we have:

$$z_t = \alpha b_{4+t} + \beta b_4 b_{38} + \omega,$$

where α, β and ω can be 0 or 1, respectively.

When $\alpha = 0$, $\beta = 1$, and $\omega + z = 1$, two NFSR sequence bit will be recovered. When $\alpha = 1$ and $\beta = 0$, an NFSR initial state bit will be recovered. Finally, when $\alpha = 1$, $\beta = 1$, and $\omega + z = 1$, two NFSR sequence bits will be recovered.

This can be used for simple partial state recovery of the NFSR. The remaining stages of the 120 NFSR sequence can then be found through exhaustive search. An estimate of the average exhaustive search requirement for modified Plantlet is provided in Table 3 of Section 6.2.

6 Experimental simulations

We have performed computer simulations of our divide and conquer attack, applying it to our modified version of Plantlet, to demonstrate proof of concept. The details of the simulation setup and results are provided in the following sections. We also provide experimental results in Section 6.2 for the partial NFSR sequence recovery of Plantlet; this is possible because of the low time complexity required to partially recover. The details for the structure of modified Plantlet are provided in Section 5.

6.1 Experimental approach

For each simulation, a random key together with random NFSR and LFSR states were produced. Output from modified Plantlet was then produced. The attack from Section 4 was then applied. The remaining NFSR sequence bits were then exhaustively searched. Each sequence candidate was used to produce output, which was checked against the correct output sequence. A candidate that produced the correct output was considered the correct state sequence state. Using this sequence, the key was recovered. The computed key, NFSR and LFSR was then checked against the correct key, NFSR and LFSR.

The code used for the simulations was written using the SageMath software package [25] and all calculations were performed using QUT's High Performance Computing facility. We used a single node from the cluster with an Intel Xeon core capable of 271 TeraFlops.

6.2 Results on modified Plantlet

In precomputation, the initial system of equations was built, the linear dependency was found and the reduced system of equations was built. For modified Plantlet, approximately 2^{20} bits of output were used. The majority of the computational complexity required for the precomputation comes from applying the linear dependency to produce the reduced system of equations. On average, precomputation was completed in 10 hours.

A total of 10 simulations were performed. In every simulation the full LFSR initial state was recovered. Each simulation for the modified version required on average 30 seconds to recover the LFSR initial state.

For each trial, partially recovering the required 120-bit NFSR sequence took approximately 2.5 hours. Table 3 provides a tally (across the 10 simulations) of how many times a certain number of state bits were recovered from the NFSR sequence. For each simulation, the full available keystream was used. That is, the NFSR sequence was partially recovered using 2^{19} bits of keystream. We see from Table 3 that on average, 67 bits were recovered for the NFSR sequence.

The remaining 53 bits required to complete the 120-bit NFSR sequence could then be recovered by exhaustive search and used to produce output. Recovery of these remaining NFSR bits would then allow the key to be determined. This portion of the simulation was not performed due to limited resources.

7 Discussion

Our experimental simulations support the theoretical model for our key recovery attack on modified Plantlet. The pre-computation stage of the

Table 3. Distribution table for NFSR sequence bits recovered over 120-bit windows using 100 simulations for modified Plantlet.

No. bits recovered	**0**	. . .	**64**	**65**	**66**	**67**	**68**	**69**	. . .	**120**
Frequency	0	. . .	0	1	4	2	2	1	. . .	0

algebraic attack is essential for recovery of the LFSR state and is the most time-consuming part of the process, but only needs to be done once. This took ten hours in our trial. In the online phase, a divide and conquer approach targeting the LFSR achieved complete recovery of the LFSR state in approximately 30 seconds with 100% success. Following this, we used a 120-bit sliding window on the NFSR state and partially recovered this window; on average we recovered 67 of these 120 bits. Guessing the remaining 53 bits and checking for consistency with observed keystream then allowed us to recover the key bits. The complexity of this guess and determine stage dominates the attack complexity but is clearly significantly less complex than exhaustive key search.

The use of key injection in the design made it possible to perform this key recovery attack without requiring any consideration of the initialisation process. That is, the initialisation is irrelevant to the security provision against these algebraic attacks. In fact, security against these attacks depends solely on the combination of the selected output function and the positions of its input bits within the two registers. This answers the open question from [1] of whether key injection provides increased security to enable reduced register sizes for lightweight stream cipher designs. While this approach may help avoid time-memory trade-off attacks, the generic structure is not robust: other attacks are possible, as we have shown.

Based on the nature of our successful attack on modified Plantlet, the following design guideline is seen to be important to the security of any keystream generator which uses a Grain-like structure with key injection:

- The output function for any such cipher should contain multiple bits taken linearly from the NFSR, with none of these bits involved in nonlinear terms of the output function that also contain bits from the LFSR.

Note that this is precisely the feature which protects the existing version of Plantlet from our attack. This guideline may need to be expanded if other types of attack on this structure are also found to be successful.

The output function of Sprout is identical to that of Plantlet, so the published version of Sprout is also protected against our attack. However, a similar modification of this output function would again allow our attack to recover the initial contents of Sprout's LFSR. But recovering the NFSR initial state and the key is more complex for Sprout, since the key bits of Sprout are fed into the NFSR conditionally. A quick estimate based on our experimental results for Plantlet suggests that the exhaustive search cost after partial NFSR recovery in this case would exceed the cost of exhaustively searching the key bits, making this attack unviable.

8 Conclusion

In this paper, we have considered the security of keystream generators using a Grain-like structure with key injection. From the above discussion, it is clear that the security of keystream generators using this design depends critically on the choice of the output function during keystream generation. Designers who employ this design approach should pay careful attention to the combination of the function used and the location of the input taps which feed it.

Ciphers of this type were designed specifically to avoid time-memory trade-off attacks, but they are not necessarily secure against other emerging attacks, such as the algebraic attack we have demonstrated here. In itself, this structure cannot be considered to provide a robust generic design for lightweight keystream generators. Indeed, the designer of a new cipher must be cautious about security implications when adding components to existing structures and should evaluate the modified design against all possible types of attacks.

A Appendix: Algorithms

A.1 Algorithm for NLFG algebraic attack

Precomputation phase:

Step 1 Use $f(S_0) = y_0$ to relate initial state bits $(s_0, s_1, \ldots, s_{n-1})$ to observed output bit y_0.

Step 2 Multiply f by a function h (if applicable) to reduce overall degree to d.

Step 3 Clock forward using $f(S_t) = y_t$ to build a system of equations of constant algebraic degree, applying the linear update as required.

Online phase:

Step 4 Substitute observed output bits $\{y_t\}_{t=0}^{\infty}$ into the system of equations.
Step 5 Solve the system of equations by linearisation, to recover $S_0 = s_0, s_1, \ldots, s_{n-1}$.

In the online phase of this attack, the initial state of the LFSR can be recovered if approximately $\binom{n}{d}$ bits of output are known. The attack has a computational complexity of $\mathcal{O}(n\binom{n}{d} + \binom{n}{d}^{\omega})$, where d is the degree of the system and ω is the Guassian elimination exponent $\omega \approx 2.8$ [7]. If the output requirement cannot be met, it may be possible to solve the system by applying other methods for solving simultaneous equations, such as Gröbner bases or the XL algorithm [5].

A.2 Algorithm for Fast algebraic attack

The precomputation phase is similar to a regular algebraic attack, with Step 3 replaced by three steps (3a, 3b and 3c) as follows.

Step 3a Identify the combination of equations that will eliminate monomials of degree e to d in the initial state bits.
Step 3b Use this linear dependency to build a new general equation.
Step 3c Use this general equation to build a system of equations of degree e in the initial state bits.

The online phase is identical to the online phase of a regular algebraic attack (but with reduced complexity).

When the Berlekamp-Massey algorithm is used to find the linear dependency, the pre-computation phase of the attack has a computational complexity of $\mathcal{O}(\binom{n}{d}log(\binom{n}{d}))$ [6]. The initial state of the LFSR can be recovered in the online phase of the attack by observing approximately $\binom{n}{d}$ bits of output with a computational complexity of $\mathcal{O}(\binom{n}{d}\binom{n}{e} + \binom{n}{e}^{\omega})$, where d is the degree of fh, e is the degree of h and $\omega \approx 2.8$ [6].

Note that at first glance the online complexities for an algebraic attack and a fast algebraic attack look similar. However, when n is much larger than d, as is the case with registers used in practice, $\binom{n}{d}^{\omega}$ is much larger than $\binom{n}{d}\binom{n}{e}$ and $\binom{n}{e}^{\omega}$. Thus, by reducing the degree from d to e, the complexity of the online phase is drastically reduced for registers of practical size.

A.3 Algorithm for LF-NFSR algebraic attack

Precomputation phase:

Step 1 A system of equations is developed using the linear filter function to represent every state bit as a linear combination of a subset of the initial state bits and some output bits. We denote this system of equation by system $\mathcal{L}$.

Step 2 A second system of equations is developed using the nonlinear update function g to represent update bits as a nonlinear combination of a subset of initial state bits. We denote this system by system $\mathcal{G}$. Substitutions are made for state bits in system $\mathcal{G}$ using system $\mathcal{L}$ where applicable to reduce the number of unknown state variables while keeping the degree of system $\mathcal{G}$ constant.

Step 3 The two systems are combined by aligning the equations from each system that represent the same state bit. The resulting system contains only initial state bits and observed output bits. We denote this system as system $\mathcal{L} + \mathcal{G}$.

Online phase:

Step 4 Substitute observed output bits $\{y_t\}_{t=0}^{\infty}$ into the system of equations

Step 5 Solve the system of equations by linearisation.

For certain update functions a reduction function of g, say h, may be used to reduce the overall degree of the system. If the degree of gh is d, then the overall system will be of degree at most d. The initial state of the LF-NFSR can be recovered in the online phase of the attack by observing approximately $\binom{n}{d}$ bits of output with a computational complexity of $\mathcal{O}(n\binom{n}{d} + \binom{n}{d}^{\omega})$, where d is the degree of the system and $\omega \approx 2.8$ [3]. Note that fast algebraic techniques are not applicable to LF-NFSRs.

References

1. Armknecht, F., Mikhalev, V.: On lightweight stream ciphers with shorter internal states. In: International Workshop on Fast Software Encryption. pp. 451–470. Springer (2015)
2. Beighton, M., Bartlett, H., Simpson, L., Wong, K.K.H.: Algebraic attacks on Grain-like keystream generators. In: International Conference of Information Security and Cryptography – ICISC 2021. pp. 241–270. Springer, Cham (2022)
3. Berbain, C., Gilbert, H., Joux, A.: Algebraic and correlation attacks against linearly filtered non linear feedback shift registers. In: International Workshop on Selected Areas in Cryptography. pp. 184–198. Springer (2008)
4. Berbain, C., Gilbert, H., Maximov, A.: Cryptanalysis of Grain. In: International Workshop on Fast Software Encryption. pp. 15–29. Springer (2006)

5. Courtois, N.T.: Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt. In: International Conference on Information Security and Cryptology. pp. 182–199. Springer (2002)
6. Courtois, N.T.: Fast algebraic attacks on stream ciphers with linear feedback. In: Annual International Cryptology Conference. pp. 176–194. Springer (2003)
7. Courtois, N.T., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 345–359. Springer (2003)
8. Englund, H., Johansson, T.: A new simple technique to attack filter generators and related ciphers. In: International Workshop on Selected Areas in Cryptography. pp. 39–53. Springer (2004)
9. Faugere, J.C., Ars, G.: An algebraic cryptanalysis of nonlinear filter generators using Gröbner bases. Report, INRIA (2003)
10. Forré, R.: A fast correlation attack on nonlinearly feedforward filtered shift-register sequences. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 586–595. Springer (1989)
11. Gammel, B.M., Göttfert, R.: Linear filtering of nonlinear shift-register sequences. In: International Workshop on Coding and Cryptography. pp. 354–370. Springer (2005)
12. Golić, J.D., Salmasizadeh, M., Simpson, L., Dawson, E.: Fast correlation attacks on nonlinear filter generators. Information Processing Letters **64**(1), 37–42 (1997)
13. Hell, M., Johansson, T., Maximov, A., Meier, W.: The Grain family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Deisgns. vol. 4986, pp. 179–190. LNCS (2006)
14. Hell, M., Johansson, T., Maximov, A., Meier, W.: A stream cipher proposal: Grain-128. In: 2006 IEEE International Symposium on Information Theory. pp. 1614–1618. IEEE (2006)
15. Hell, M., Johansson, T., Meier, W.: Grain: a stream cipher for constrained environments. International Journal of Wireless and Mobile Computing **2**(1), 86–93 (2005)
16. Hell, M., Johansson, T., Meier, W.: Grain-128a: a new version of Grain-128 with optional authentication. International Journal of Wireless and Mobile Computing **5**, 48–59 (2011)
17. Hell, M., Johansson, T., Meier, W., Sönnerup, J., Yoshida, H.: Grain-128AEAD - a lightweight AEAD stream cipher. NIST Lightweight Cryptography Competition (2019), https://csrc.nist.gov/Projects/lightweight-cryptography/finalists
18. Hong, J., Sarkar, P.: New applications of time memory data tradeoffs. In: Roy, B. (ed.) Advances in Cryptology - ASIACRYPT 2005 (LNCS vol. 3788). pp. 353–372. Springer Berlin Heidelberg" (2005)
19. Katz, J., Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC press (1996)
20. Massey, J.: Shift-register synthesis and BCH decoding. IEEE Transactions on Information Theory **15**(1), 122–127 (1969)
21. Meier, W., Staffelbach, O.: Fast correlation attacks on certain stream ciphers. Journal of Cryptology **1**(3), 159–176 (1989)
22. Mikhalev, V., Armknecht, F., Müller, C.: On ciphers that continuously access the non-volatile key. IACR Transactions on Symmetric Cryptology pp. 52–79 (2016)
23. Millan, W.: Analysis and Design of Boolean Functions for Cryptographic Applications. PhD Thesis, Queensland University of Technology (1997)

24. Siegenthaler, T.: Cryptanalysts representation of nonlinearly filtered ML-sequences. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 103–110. Springer (1985)
25. Stein, W., Joyner, D.: Sage: System for algebra and geometry experimentation. ACM Bulletin **39**(2), 61–64 (2005)