# Exploiting Transformations of the Galois Configuration to Improve Guess-and-Determine Attacks on NFSRs

Gefei Li[1], Yuval Yarom[1], and Damith C. Ranasinghe[1]

University of Adelaide, Adelaide, South Australia, Australia
gefei.li@student.adelaide.edu.au,
yval@cs.adelaide.edu.au,
damith@cs.adelaide.edu.au

**Abstract.** Guess-and-determine attacks are based on guessing a subset of internal state bits and subsequently using these guesses together with the cipher's output function to determine the value of the remaining state. These attacks have been successfully employed to break NFSR-based stream ciphers. The complexity of a guess-and-determine attack is directly related to the number of state bits used in the output function. Consequently, an opportunity exits for efficient cryptanalysis of NFSR-based stream ciphers if NFSRs used can be transformed to derive an equivalent stream cipher with a simplified output function.
In this paper, we present a new technique for transforming NFSRs. We show how we can use this technique to transform NFSRs to equivalent NFSRs with simplified output functions. We explain how such transformations can assist in cryptanalysis of NFSR-based ciphers and demonstrate the application of the technique to successfully cryptanalyse the lightweight cipher Sprout. Our attack on Sprout has a time complexity of $2^{70.87}$, which is $2^{3.64}$ times better than any published non-TMD attack, and requires only 164 bits of plaintext-ciphertext pairs.

**Keywords:** Guess-and-determine, NFSR, Sprout

## 1 Introduction

Stream ciphers generate a sequence of pseudorandom bits called a *keystream* which is combined with plaintext to produce a stream of ciphertext. Stream ciphers are attractive because, generally, they can achieve high throughput and their simplicity makes them suitable for resource constrained devices such as wireless sensors and Radio Frequency Identification tags. A typical stream cipher for achievement of fast encryption of messages contains feedback functions and an output function [28]. Linear feedback shift registers (LFSRs) are often used in the design of stream ciphers because LFSRs are simple and easy to implement in hardware. However, they are insecure against several attacks such as fast correlation attack [24], and algebraic attack [10, 25]. Nonlinear feedback shift registers (NFSRs) are introduced as a protection against these attacks.

There are two ways to implement NFSRs. An $n$-bit NFSR in the *Fibonacci* configuration (Figure 1), has one feedback function which updates the most significant bit (bit $n-1$) of the register. The values of other bits are determined by shifting. That is, bit $i$ is shifted to the bit $i-1$ ($i = n-1, ..., 1$).

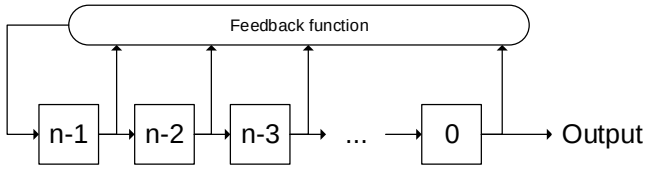


Fig. 1: The Fibonacci configuration of NFSRs

In the *Galois* configuration of NFSRs (Figure 2), each bit is updated using its own feedback function. Therefore, there are up to $n$ feedback functions.

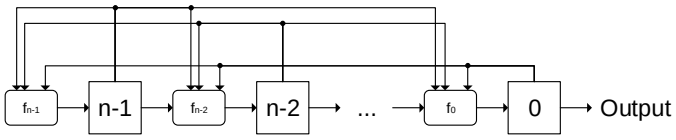


Fig. 2: The Galois configuration of NFSRs

Dubrova [11] has formulated a transformation of the Fibonacci configuration to the Galois configuration for NFSRs with the aim of increasing the computation speed. She later outlined a method for deriving the corresponding initial states to generate identical output keystreams [12]. Dubrova [14] further elaborates transformations for NFSRs. In [11, 14], Dubrova identifies several criteria that should be satisfied in order to maintain an equivalent output keystream sequence before and after the transformation of the stream cipher. Dubrova [14] notes the potential for exploiting NFSR transformations for cryptanalysis, however no such use is proposed and to the best of our knowledge this concept has, so far, not been used for cryptanalysis.

The basic idea of guess-and-determine attack (GD attack) is to guess a part of the internal state of the cipher, and use the guessed state to determine the value of other unknown state. The attack was first introduced by Knudsen et al. [21] and by Bleichenbacher and Patel [9] and was later named "guess-

and-determine" [15, 19]. Since then, the guess-and-determine attack has been used to target several stream ciphers [1, 17, 27, 29, 30].

We propose an improved guess-and-determine attack against NFSR-based stream ciphers. The attack requires several rounds to guess and determine internal state bits involved in the output function of the stream cipher. At each round, one bit of the keystream sequence is used to determine one bit of the internal state. Meanwhile, the output functions used in other clocks feedback functions of the stream ciphers are used to create Boolean functions system. More information collected from the keystream sequence is able to reduce the complexity of the attack and also increase the number of rounds we need. Consequently, to reduce the cardinality of the dependence set of the output function of stream ciphers, the techniques of transformation on NFSRs are adopted in this attack. However, previous work on transformation on NFSRs contains several criteria on transformation. We develop the theorem on transformation on NFSRs as well.

In FSE 2015, Armknecht and Mikhalev [3] proposed a lightweight stream cipher called Sprout. The main inspiration of the design is to resist the time-memory-data (TMD) tradeoff attacks [4, 6–8, 18], and break the rule of thumb that the size of internal state should be at least twice the size of key. To achieve these goals, the key bits are treated as implied internal state and are included in the feedback function. This brings the total number of expressed and implied internal state bits to twice the size of key. While only small part of key bits involved in the feedback function of Sprout. In cryptanalytic perspective, by guessing a small size of the key bits, the attackers are able to recover the whole expressed internal states.

Lallemand and Naya-Plasencia [22] apply a rebound attack [26] with a time complexity of $2^{74.51}$, which is $2^{10}$ times faster than a brute force attack. (The attack requires the equivalent of $2^{69.36}$ tries of a brute force attack, compared with $2^{80}$ required for the full brute force attack.) This attack, as well as other attacks on Sprout, are summarised in Table 1.

Table 1: Summary of attacks on Sprout

| Type of attack | Data | Time | Memory |
|---|---|---|---|
| Rebound attack [22] | 112 | $2^{74.51}$ | $2^{46}$ |
| TMD tradeoff attack[16] | $2^{40}$ | $2^{45}$ | 770 terabytes |
| Distinguishing attack[5] | | $2^{75}$ | $O(n)$ |
| Guess-and-determine attack (this paper) | | $2^{70.87}$ | $O(n)$ |

The main contribution of this paper is showing how to exploit NFSR transformations for cryptanalysis, in particular for reducing the time complexity of guess-and-determine attacks. Firstly, we generalise the work of Dubrova [14] by relaxing the limitations on NFSR transformations. We prove that our transformations result in equivalent NFSRs, producing the same output sequence. We then show how to use these transformations to reduce the complexity of the

NFSR output function, and with it the complexity of a guess-and-determine attacks against the NFSR. Finally, we apply the transformation to the NFSR used in the Sprout cipher and implement a guess-and-determine attack on the transformed cipher. Our attack has a complexity of $2^{70.87}$, which is $2^{3.64}$ times better than the best known non-TMD attack [22].

The paper is organised as follows: Section 2 provides background on NFSRs and their transformations and introduces the notation used in this paper. Section 3 presents our new transformations of NFSRs and proves that they preserve the output sequence. Section 4 shows how to use the transformations to improve guess-and-determine attacks against NFSR-based ciphers. In Section 5 we apply the technique to the Sprout cipher.

## 2 Preliminaries

### 2.1 Boolean functions

We first introduce some definitions and notations we use throughout the paper. These are, mostly, based on the notation of Dubrova [11].

Let $GF(2)$ be the binary field with $\oplus$ denoting the addition operation and juxtaposition denoting the field multiplication. For an arbitrary $n$, $GF^n(2)$ is the $n$-dimensional vector space over $GF(2)$. A *Boolean function* is a function $f : GF^n(2) \to GF(2)$. We sometimes use $f(x)$ to replace the explicit notation $f(x_0, \ldots, x_{n-1})$ when the number of variables is clear from the context.

A *product term* of the variables $x_0, \ldots, x_{n-1}$ is a term of the form $x^I = \prod_{i \in I} x_i$ for some $I \subseteq \{0, \ldots, n-1\}$.

Any Boolean function can be written as the sum of product terms of its input variables—a format called the *algebraic normal form* (ANF). More formally,

$$f(x_0, ..., x_{n-1}) = \bigoplus_{I \subseteq \{0,\ldots,n-1\}} c_I x^I$$

where $c_I \in \{0, 1\}$ is called the *coefficient* of the product term $x^I$.

The *dependence set* of a Boolean function $f$ is the set of those input variables that affect the function's value. That is, $i \in dep(f)$ iff there exist $x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{n-1}$ such that $f(x_0, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_{n-1}) \neq f(x_0, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_{n-1})$ We note that for a product term we always have $dep(x^I) = I$.

Product terms for which the cardinality of the dependence set is one, such as $x_0$ or $x_{15}$, are called *linear product terms*. Product terms with a larger cardinality of the dependence set, for example $x_1x_2$, are *non-linear product terms*. The *affine* product term $x^\emptyset = 1$ is neither linear nor non-linear. We do not use the affine product term in this work.

For a Boolean function $f(x_0, \ldots, x_{n-1})$ we define the positive and negative $m$-shifts of $f$, denoted by $f|_{+m}$ and $f|_{-m}$, as the function obtained from $f$ by increasing and decreasing the indices of the input variables by $m$. That is,

$$f|_{+m}(x_0, \ldots, x_{n-1}) = f(x_m, \ldots, x_{n-1}, x_0, \ldots, x_{m-1})$$
$$f|_{-m}(x_0, \ldots, x_{n-1}) = f(x_{n-m}, \ldots, x_{n-1}, x_0, \ldots, x_{n-m-1})$$

### 2.2 Feedback Shift Registers

A *shift register* is a storage element consisting of $n$ bits, known as the *state* of the register. We use the notation $s_i^t$ for the value of bit $i$ of the state at time $t$, and collectively denote the whole state at time $t$ by $s^t = (s_0^t, \ldots, s_{n-1}^t)$. At each round, each of the state bits receives the value of the next bit, i.e. $s_i^t = s_{i+1}^{t-1}$ for $0 \leq i < n-1$. At time $t$ the register gets an input which is entered into $s_{n-1}^t$. The output of the register at time $t$ is the value $s_0^t$.

In a simple form of a *feedback shift register*, the input is calculated using a Boolean *feedback function* $f_{n-1}$ of the state. That is, $s_{n-1}^t = f_{n-1}(s^{t-1}) = f_{n-1}(s_0^{t-1}, \ldots, s_{n-1}^{t-1})$. This form of feedback shift registers is usually called the *Fibonacci* configuration.

A more generalised form, known as the *Galois* configuration, has feedback functions feeding into each bit. More formally, $s_i^t = f_i(s^{t-1})$. A feedback functions of the type $f_i(s^t) = s_{i+1}^{t-1} \oplus g_i(s^{t-1})$ such that $i+1 \notin dep(g_i)$ is called *singular*. Note that index arithmetic is done modulo-$n$, i.e. $s_n^t$ is the same as $s_0^t$ and $n \in dep(g_i)$ iff $0 \in dep(g_i)$.

Further generalisations of feedback shift registers include combining external inputs with the feedback functions or using a Boolean *output function* $z(s^t)$ to generate the output at time $t$. Two feedback shift registers are considered *equivalent* if they produce the same output sequence when given the same input sequence.

In a *linear feedback shift register* (LFSR) the feedback functions and the output function are all linear. Otherwise the register is a *non-linear feedback shift register* (NFSR).

### 2.3 Transformations of NFSRs

Several works have looked at conversions between Galois and Fibonacci NFSRs. Dubrova [11] looks at converting Fibonacci NFSRs to the Galois configuration with the aim of improving the performance by reducing the depth of the logical circuit used for implementing the feedback function. Dubrova notices that under certain conditions, an NFSR can be converted to an equivalent NFSR by shifting product terms between the feedback functions of the bits of the NFSR. More specifically, the shifts are possible if the NFSRs are *uniform*. That is, they are split around a *terminal* bit $\tau$, such that all of the non-trivial feedback functions update bits above the terminal bit and take their input from state bits below the terminal bit.

Dubrova [12] shows how to update the initial state of the NFSR to ensure that the output sequence is the same after the shift of product terms when using a trivial output function $z(t) = s_0^t$.

Dubrova [14] specifies a set of constraints for shifting product terms between feedback functions. The constraints are the same as those we present in Section 3.1. These constraints are more relaxed than the requirement of using uniform NFSRs.

Lin [23] examines the equivalence between the Galois and the Fibonacci configurations of NFSRs demonstrating that Galois NFSRs of a specific construction can be converted into equivalent Fibonacci NFSRs. The construction requires that the feedback functions only take input from lower bits than the bits they update. That is, $s_i^{t+1} = s_{i+1}^t \oplus g_i(s^t)$, such that $dep(g_i) \subset \{0, \ldots, i-1\}$[1]. For these NFSRs, Lin [23] iteratively replaces occurrences of $s_i^t$ in the feedback functions with the expanded form $s_i^t = s_{i+1}^{t-1} \oplus g_i(s_0^{t-1}, \ldots, s_{i-1}^{t-1})$. Each such replacement effectively shifts the non-trivial part of the feedback function of bit $i$, creating an equivalent NFSR in which the feedback function of bit $i$ is trivial, i.e. $s_i^{t+1} = s_{i-1}^t$. The resulting NFSR at the end of the process has only one non-trivial feedback function and is, therefore, a Fibonacci NFSR.

## 3 A Generalised NFSR Transformation

In this section we present a generalisation of the transformations suggested in previous works. For simplicity, we look at transformations that shift a single product term one bit down the NFSR. Equivalent transformations that shift the product term up one bit can be constructed in a similar manner. We note that by repeatedly shifting multiple product terms we can achieve the shifts suggested in the prior works.

### 3.1 The Basic Shift

We have an $n$-bit NFSR $N$, with feedback functions $f_i(x) = x_{i+1} \oplus g_i(x)$ and an output function $z(x)$. We want to shift a product term $p = x^I$, for some $I \subseteq \{0, \ldots, n-1\}$, from the feedback function $f_\alpha$ to $f_{\alpha-1}$.

Intuitively, what we want is to create an equivalent NFSR $\bar{N}$ with feedback functions $\bar{f}_i(x) = x_{i+1} \oplus \bar{g}_i(x)$ and an output function $\bar{z}(x)$ such that the main difference between $N$ and $\bar{N}$ is that $\bar{f}_\alpha(x) = f_\alpha(x) \oplus p(x)$ and $\bar{f}_{\alpha-1}(x) = f_{\alpha-1} \oplus p|_{-1}(x)$. In the general case, we expect to need to have more differences between $\bar{N}$ and $N$ to achieve equivalence. However, as a warmup, we first look at a restricted scenario, in which $\bar{N}$ does not require any further changes. The constraints we have are:

1. $\alpha + 1 \notin I$.
2. $f_{i-1}(x) = x_i$ for all $i \in I$.
3. $\alpha \notin dep(g_i)$ for any $i \neq \alpha$.
4. $\alpha \notin dep(z)$.

As mentioned above, these constraints are the same as the constraints of Dubrova [14]. The main difference between the setting we present here and the settings of Dubrova is that we only shift the product term to a neighbouring bit, hence we require multiple shifts to achieve the results of Dubrova.

[1] Lin [23] defines Galois NFSRs as those having this specific construction. This is a subset of the Galois configuration as the term is used in Dubrova [11] and in this paper.

We now define a *shift state transform* that we will use for showing that $\bar{N}$ and $N$ are equivalent.

**Definition 1.** *Let $p = x^I$ be a product term, and $0 \leq \alpha < n$. The shift state transform $T_{p,\alpha} : GF^n(2) \to GF^n(2)$ is defined as follows:*

$$T_{p,\alpha}(s_0, \ldots, s_{n-1}) = (s_0, \ldots, s_{\alpha-1}, s_\alpha \oplus p|_{-1}(s_0, \ldots, s_{n-1}), s_{\alpha+1}, \ldots, s_{n-1})$$

**Lemma 1.** *Let $s^t$ and $\bar{s}^t$ denote the states of $N$ and $\bar{N}$, respectively, at time $t$. If $\bar{s}^t = T_{p,\alpha}(s^t)$ then $\bar{s}^{t+1} = T_{p,\alpha}(s^{t+1})$.*

*Proof.* We first note that due to Constraint 2 if $i \in I$, $s_i^{t+1} = s_{i+1}^t$. Hence, we have $p(s^t) = p|_{-1}(s^{t+1})$. We also note that from Constraint 1 we have $\alpha+1 \notin dep(p)$, which implies $\alpha \notin dep(p|_{-1})$. Because the only difference between $s^t$ and $\bar{s}^t$ is in bit $\alpha$ we have $p|_{-1}(s^t) = p|_{-1}(\bar{s}^t)$ and also, by Constraint 3, we have $g_i(\bar{s}^t) = g_i(s^t)$.

We now looks at the bits of state $s^{t+1}$. For $i \neq \alpha, i \neq \alpha - 1$ we have $\bar{g}_i = g_i$. Consequently

$$\bar{s}_i^{t+1} = \bar{s}_{i+1}^t \oplus \bar{g}_i(\bar{s}^t) = s_{i+1}^t \oplus g_i(\bar{s}^t) = s_i^{t+1}$$

For state bit $\alpha - 1$.

$$\begin{aligned}
\bar{s}_{\alpha-1}^{t+1} &= \bar{f}_{\alpha-1}(\bar{s}^t) \\
&= f_{\alpha-1}(\bar{s}^t) \oplus p|_{-1}(\bar{s}^t) \\
&= \bar{s}_\alpha^t \oplus g_{\alpha-1}(\bar{s}^t) \oplus p|_{-1}(\bar{s}^t) \\
&= s_\alpha^t \oplus p|_{-1}(\bar{s}^t) \oplus g_{\alpha-1}(\bar{s}^t) \oplus p|_{-1}(\bar{s}^t) \\
&= s_\alpha^t \oplus g_{\alpha-1}(\bar{s}^t) \\
&= s_\alpha^t \oplus g_{\alpha-1}(s^t) \\
&= s_{\alpha-1}^{t+1}
\end{aligned}$$

For bit $\alpha$ we have:

$$\begin{aligned}
\bar{s}_\alpha^{t+1} = \bar{f}_\alpha(\bar{s}^t) = f_\alpha(\bar{s}^t) \oplus p(\bar{s}^t) = f_\alpha(s^t) \oplus p(\bar{s}^t) \\
= f_\alpha(s^t) \oplus p|_{-1}(\bar{s}^{t+1}) = f_\alpha(s^t) \oplus p|_{-1}(s^{t+1})
\end{aligned}$$

Thus

$$\begin{aligned}
\bar{s}^{t+1} &= (\bar{s}_0^{t+1}, \ldots, \bar{s}_{n-1}^{t+1}) \\
&= (s_0^{t+1}, \ldots, s_{\alpha-1}^{t+1}, s_\alpha^{t+1} \oplus p|_{-1}(s^{t+1}), s_{\alpha+1}^{t+1}, \ldots, s_{n-1}^{t+1}) \\
&= T_{p,\alpha}(s^{t+1})
\end{aligned}$$

□

**Theorem 1.** *If $\bar{s}^0 = T_{p,\alpha}(s^0)$ then for all $t$, $\bar{s}^t = T_{p,\alpha}(s^t)$.*

*Proof.* Follows from Lemma 1 by induction on $t$. □

**Theorem 2.** *$N$ and $\bar{N}$ are equivalent.*

*Proof.* By definition, $\bar{z}$, the output function of $\bar{N}$ is the same as the output function of $N$. We need to show that for any initial state of $N$ we can find an initial state of $\bar{N}$ such that the output sequences of the two NFSRs are the same. Let $s^0$ be the initial state of $N$. We show that if $\bar{N}$ is initialised to $T_{p,\alpha}(s^0)$ then the output sequences are the same.

By Theorem 1 we know that for any $t$, $\bar{s}^t = T_{p,\alpha}(s^t)$. Hence, for any $t$, and for any $i \neq \alpha$, $\bar{s}_i^t = s_i^t$. From Constraint 4 we know that $z(s^t)$ does not depend on the value of $s_\alpha^t$. Hence, $\bar{z}(\bar{s}^t) = z(\bar{s}^t) = z(s^t)$. □

We note that the operations we use in the transform are all reversible, hence a similar construction can be used to shift $p$ in the reverse direction, i.e. from $f_\alpha$ to $f_{\alpha+1}$

### 3.2 Relaxing the Constraints

In Section 3.1 we specified several constraints on the shifted product term and used those to establish an invariant, based on the shift state transform of Definition 1. In this section we want to relax some of the constraints, namely Constraints 3 and 4, while preserving the invariant.

The idea is simple: if we assume that the invariant holds we can modify the feedback and output functions of $\bar{N}$ to account for the modified state value $\bar{s}_\alpha^t$. That is, the feedback function we use is such that $\bar{f}(x) = f(T_{p,\alpha}^{-1}(x))$, and because $T_{p,\alpha} = T_{p,\alpha}^{-1}$, we have $\bar{f}(x) = f(T_{p,\alpha}(x))$. The following theorem applies this idea more formally.

**Theorem 3.** *Let $N$ be an $n$-bit NFSR, with feedback functions $f_i(x) = x_{i+1} \oplus g_i(x)$, and an output function $z(x)$. Let $p = x^I$ be a product term such that Constraints 1 and 2 above hold. If $\bar{N}$ is an $n$-bit NFSR, with feedback functions $\bar{f}_i(x)$ and an output function $\bar{z}(x)$ defined as follows:*

$$\bar{f}_i(x) = \begin{cases} x_\alpha \oplus g_{\alpha-1}(T_{p,\alpha}(x)) \oplus p|_{-1}(x) & i = \alpha - 1 \\ f_\alpha(T_{p,\alpha}(x)) \oplus p(x) & i = \alpha \\ f_i(T_{p,\alpha}(x)) & \textit{Otherwise} \end{cases}$$
$$\bar{z}(x) = z(T_{p,\alpha}(x))$$

*then $N$ and $\bar{N}$ are equivalent.*

*Proof.* We first show that the invariant holds, i.e. that if $\bar{s}^t = T_{p,\alpha}(s^t)$ then $\bar{s}^{t+1} = T_{p,\alpha}(s^{t+1})$. For $i \neq \alpha - 1, i \neq \alpha$ we have:

$$\bar{s}_i^{t+1} = \bar{f}_i(\bar{s}^t) = f_i(T_{p,\alpha}(\bar{s}^t)) = f_i(s^t) = s_i^{t+1}$$

For state bit $\alpha - 1$ we have:

$$\begin{aligned}
\bar{s}_{\alpha-1}^{t+1} &= \bar{f}_{\alpha-1}(\bar{s}^t) \\
&= \bar{s}_{\alpha}^{t} \oplus g_{\alpha-1}(T_{p,\alpha}(\bar{s}^t)) \oplus p|_{-1}(\bar{s}^t) \\
&= s_{\alpha-1}^{t} \oplus p|_{-1}(s^t) \oplus g_{\alpha-1}(T_{p,\alpha}(\bar{s}^t)) \oplus p|_{-1}(\bar{s}^t) \\
&= s_{\alpha-1}^{t} \oplus g_{\alpha-1}(T_{p,\alpha}(\bar{s}^t)) \\
&= s_{\alpha-1}^{t} \oplus g_{\alpha-1}(s^t) \\
&= s_{\alpha-1}^{t+1}
\end{aligned}$$

Finally, for state bit $\alpha$:

$$\bar{s}_{\alpha}^{t+1} = \bar{f}_{\alpha}(\bar{s}^t) = f_{\alpha}(T_{p,\alpha}(\bar{s}^t)) \oplus p(\bar{s}^t) = f_{\alpha}(s^t) \oplus p(\bar{s}^t) = f_{\alpha}(s^t) \oplus p|_{-1}(\bar{s}^{t+1}) = f_{\alpha}(s^t) \oplus p|_{-1}(s^{t+1})$$

Thus,

$$\begin{aligned}
\bar{s}^{t+1} &= (\bar{s}_0^{t+1}, \ldots, \bar{s}_{n-1}^{t+1}) \\
&= (s_0^{t+1}, \ldots, s_{\alpha-1}^{t+1}, s_{\alpha}^{t+1} \oplus p|_{-1}(s^{t+1}), s_{\alpha+1}^{t+1}, \ldots, s_{n-1}^{t+1}) \\
&= T_{p,\alpha}(s^{t+1})
\end{aligned}$$

By induction on $t$ we show that if $\bar{s}^0 = T_{p,\alpha}(s^0)$ than for all $t$ $\bar{s}^t = T_{p,\alpha}(s^t)$ hence we have $\bar{z}(\bar{s}^t) = z(T_{p,\alpha}(\bar{s}^t)) = z(s^t)$ and the output sequences of $N$ and $\bar{N}$ are identical. □

### 3.3 Generalised Transformation

We now want to remove Constraint 2. When we remove the constraint, we can no longer claim that shifting the product term only introduces a one round delay. That is, we no longer have $p(s^t) = p|_{-1}(s^{t+1})$. Consequently, to preserve the invariant we need to update $\bar{f}_{\alpha}$. The fix, basically, reverses the feedback functions of the inputs of $p|_{-1}$. This fix is formalised in Theorem 4.

**Theorem 4.** *Let $N$ be an $n$-bit NFSR, with feedback functions $f_i(x) = x_{i+1} \oplus g_i(x)$, and an output function $z(x)$. Let $p = x^I$ be a product term such that $\alpha + 1 \notin I$. If $\bar{N}$ is an $n$-bit NFSR, with feedback functions $\bar{f}_i(x)$ and an output function $\bar{z}(x)$ defined as follows:*

$$\bar{f}_i(x) = \begin{cases} x_{\alpha} \oplus g_{\alpha-1}(T_{p,\alpha}(x)) \oplus p|_{-1}(x) & i = \alpha - 1 \\ f_{\alpha}(T_{p,\alpha}(x)) \oplus p|_{-1}(\bar{f}_0(x), \ldots, \bar{f}_{\alpha-1}(x), 0, \bar{f}_{\alpha+1}(x), \ldots, \bar{f}_{n-1}(x)) & i = \alpha \\ f_i(T_{p,\alpha}(x)) & \textit{Otherwise} \end{cases}$$

$$\bar{z}(x) = z(T_{p,\alpha}(x))$$

*then $N$ and $\bar{N}$ are equivalent.*

*Proof.* We note that the only difference from Theorem 3 is the definition of $\bar{f}_{\alpha}$, which only affects state bit $\alpha$. For this bit we now have:

$$\begin{aligned}
\bar{s}_{\alpha}^{t+1} &= \bar{f}_{\alpha}(\bar{s}^t) \\
&= f_{\alpha}(T_{p,\alpha}(\bar{s}^t)) \oplus p|_{-1}(\bar{f}_0(\bar{s}^t), \ldots, \bar{f}_{\alpha-1}(\bar{s}^t), 0, \bar{f}_{\alpha+1}(\bar{s}^t), \ldots, \bar{f}_{n-1}(\bar{s}^t)) \\
&= f_{\alpha}(s^t) \oplus p|_{-1}(s_0^{t+1}, \ldots, s_{\alpha-1}^{t+1}, 0, s_{\alpha+1}^{t+1}, \ldots, s_{n-1}^{t+1})
\end{aligned}$$

and because $\alpha \notin dep(p|_{-1})$, we have $\bar{s}_{\alpha}^{t+1} = f_{\alpha}(s^t) \oplus p|_{-1}(s^{t+1}) = s_{\alpha}^{t+1} \oplus p|_{-1}(s^{t+1})$.

The rest of the proof is the same as the proof of Theorem 3 □

### 3.4 Discussion

We can compare our work to previously suggested transformations based on two different criteria: the allowed combinations of inputs and outputs of the product terms that can be shifted and the other constraint on the shift. On the first criterion, we see that the work of Dubrova [11] and its extension [12] are limited to only move product terms that have inputs below $\tau$ and output above $\tau$. The space of possible shift is depicted in Figure 3. The diagram displays the possible inputs in the X axis and possible outputs in the Y axis, with a rectangular area corresponding to these shifts.

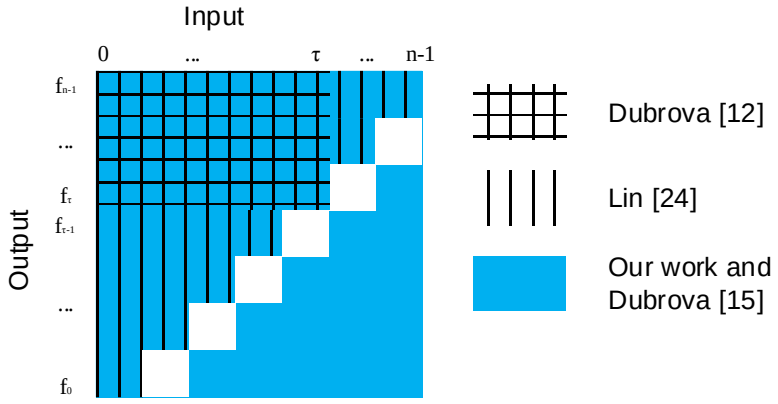


Fig. 3: The conditions to apply different methods of transformation on NFSRs

In Lin [23] the feedback function of bit $i$ depends only on state bits below $i$. This set of combinations is a superset of the possible combinations in Dubrova [11]. This set covers all the area above the main diagonal in Figure 3.

Dubrova [14] and our work increase the possible combinations. Both works can shift product terms from bit $\alpha$ as long as the product terms do not depend on bit $\alpha + 1$. The reason we cannot shift product terms from bit $\alpha$ if they depend on bit $\alpha + 1$ is that such shifts are not generally invertible. As Dubrova [14] notes, such product terms are not common in cryptography because the use of non-invertible functions may result in a loss of entropy [20].

On the second criterion, we have already mentioned that Dubrova [14] imposes the same constraint as our Theorem 2. These constraints also apply implicitly to uniform NFSRs [11, 12]. As discussed above, when all of the constraints are imposed, the only changes required for equivalence are in the feedback functions of the bit the product term is shifted from and the bit the product term is shifted to. By adjusting the initial state, we can guarantee equivalence of the output sequences.

The transformation of Lin [23] does not impose our Constraint 3. Consequently, it results in changes in feedback functions that depend on the original output of the shifted product term. The order of applying the transformations implies our Constraint 2 and the implied use of the output function $z(x) = x_0$ implies our Constraint 4. Thus, our work is the first to apply NFSR transformations without Constraints 2 and 4.

Another constraint implicit in all prior works is that they all shift product terms that exist in the ANF of the feedback function. Our transformation does not impose this constraint. Naturally, shifting a product term from a feedback function it does not exist in will add the product term to that feedback function, increasing it complexity. However, as we show below the complexity of the feedback function has little implications on the guess-and-determine attack we use.

## 4 Transforming NFSRs to Improve Guess-and-Determine Attacks

Dubrova [14] speculates that NFSR transformations may be useful for cryptanalysis, however, this speculation is not substantiated and exploring this is left for future work. In this section we show an example of how transforming an NFSR can facilitate guess-and-determine attacks. We first introduce guess-and-determine attacks, with focus on attacks in the context of NFSR-based stream ciphers and show how to apply the attack to a toy NFSR. We then explain how reducing the complexity of the output function facilitates guess-and-determine attacks, and show how transforming the toy NFSR can be reduce the complexity of the output function and with it the complexity of the attack.

### 4.1 The Guess-and-Determine Attack

A guess-and-determine attack is a type of known plaintext attack that has been successfully employed against stream ciphers [9, 15, 17, 19, 21, 29, 30]. In a known plaintext attack model the adversary has access to both the plaintext and its encrypted ciphertext and, consequently, the adversary can recover the secret keystream used for encrypting the known plaintext to the known ciphertext. The central idea of a guess-and-determine attack is to part of the internal state of a cipher and subsequently use the known keystream bits to determine the remaining state.

When applied to NFSR-based stream ciphers, the attack first guesses the values of state bits in the dependence set of the output function. It then eliminates guesses that are inconsistent with the (known) output bit. This guess and elimination steps can be viewed as guessing some state bits and using the output function to determine the value of others. If there are remaining guesses, the attack uses the NFSR feedback functions to propagate the previously guessed bits and repeats with a guess of the next round.

If there are no remaining guesses at a round, the attack backtracks to the previous round and tries another guess. If, at some stage, all of the state bits are guessed, the attack "runs" the NFSR for several rounds to ensure it is consistent with the known output stream. This general attack methodology is illustrated in algorithm 1.

**Input** : An $n$-bit keystream sequence, the output function and feedback functions of the cipher
**Output**: Internal state

Initialise the system of Boolean functions based on the output functions and feedback functions;
$i \leftarrow 0$;
**while** *System of boolean functions is not recovered* **do**
  Guess values of unknown variables in the dependent set of the output function in the $i$-th round;
  Use the $i$-th bit of the keystream to determine variable(s) in the dependence set of the output function;
  **if** *the system of Boolean functions is solved* **then**
    **return** the information from the $i$-th round to determine the values of key bits
  **end**
  **else if** *the Boolean functions system is consistent* **then**
    $i \leftarrow i + 1$;
    Move to the next round;
  **end**
  **else**
    Reject the wrong guess, and move to the next guess;
    **if** *all guesses are exhausted and there is no correct guess* **then**
      $i \leftarrow i - 1$ Move to the previous round;
    **end**
  **end**
**end**

**Algorithm 1:** Guess-and-Determine Attack

Generally, as the attack progresses, information from some of the previously guessed bits will propagate to state bits in the dependence set of the output function, reducing the number of bits to be guessed in future rounds. For example, if the output function depends on $x_i$ and $x_{i+2}$, and assuming trivial feedback

functions for state bits $i$ and $i+1$, a guess of $s_{i+2}^t$ would propagate in two rounds to $s_i^{t+2}$, hence there would be no need to guess the latter.

When the output function is non-linear, the amount of reduction in the number of bits to guess may depend on the value of previously guessed bits. For example, if the ANF of the output function contains the product term $x_i x_j$, and at round $t$ the adversary has a guess for $s_i^t$, the guessed value will determine whether $s_j^t$ needs to be guessed. If $s_i^t = 0$, the product term evaluates to 0 irrespective of the value of $s_j^t$. There is, therefore, no need to guess the latter. If, however, $s_i^t = 1$, the value of the product term depends on $s_j^t$, which needs to be guessed. Similarly, when the a feedback function is not linear, it may sometimes be possible to use it to propagate guesses even if not all of its inputs are known.

### 4.2 Attack Example

As an illustrative example we now consider a toy 6-bit NFSR $N_1$ with the following feedback functions:

$$\begin{aligned} f_5(x) &= x_0 \oplus x_1 \oplus x_2 \oplus x_1 x_2 \\ f_4(x) &= x_5 \\ f_3(x) &= x_4 \\ f_2(x) &= x_3 \\ f_1(x) &= x_2 \\ f_0(x) &= x_1 \end{aligned}$$

which has a period of 63 [13].

The output function we use is

$$z(x) = x_0 \oplus x_1 \oplus x_2 x_3 \oplus x_4 \oplus x_5$$

A brute force attack on the cipher has to guess each possible combination of the initial state of the cipher, resulting in a search space of size $2^6$.

To perform a guess-and-determine attack on $N_1$, we guess the initial state of bits 0 to 4, i.e. $s_0^0, \ldots, s_4^0$. Using the output function and the first output bit, we can now determine the value of $s_5^0$ and recover the initial state with an attack complexity of $2^5$. Figure 4 illustrates this attack.

### 4.3 Improving the Guess-and-Determine Attack

When we examine the guess-and-determine attack above, we can see that we use one bit of information from the output sequence. By using this bit we can reduce the attack complexity by one bit. To do better we will need to use more bits from the output sequence, and for that we need to be able to guess less than five bits in the first round and still be able to use the first output bit to determine parts of the state.

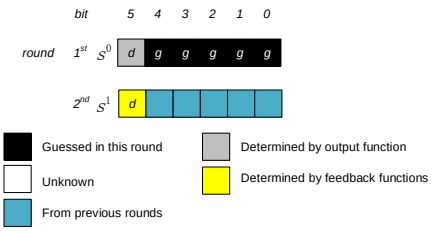


Fig. 4: The guess-and-determine attack on NFSR $N_1$

One way to achieve this is to rely on the non-linearity of the output function. For the specific example of $N_1$, we note that if we guess $s_2^0 = 0$, there is no need to guess $s_3^0$. We can use $s_0^0$, $s_1^0$, $s_2^0$, $s_4^0$ and the output bit to determine $s_5^0$.

Not guessing $s_3^0$ means that we cannot propagate its value the second round, hence we will not have the values $s_2^1$, but we will have the values of all of the other state bits. If $s_3^1 = 1$, we can use the next output bit to determine the value of $s_2^1$ and complete the attack. Otherwise, we note that $s_1^1 = 0$, hence $s_5^2$ does not depend on $s_2^1$ and we can move to the next round. In the next round we still miss the value of the last bit $s_1^2$, which we can determine from the output bit. These scenarios are illustrated in Figure 5.

In summary, in half of the cases, when we guess $s_2^0 = 1$, we need to guess 5 bits and in the other half, when $s_2^0 = 0$ we only need to guess 4 bits. This gives a total of 24 cases to search and the attack complexity is $2^{4.58}$.

### 4.4 Reducing the Complexity of the Output Function

Another way of reducing the number of state bits we need to guess in each round is to reduce the cardinality of the dependence set of the output function. To be able to use the output function to determine state bit values, we need to guess the values of all but one of the inputs of the output function. Reducing the cardinality of the output function reduces the number of bits we need to guess each round and has the potential of increasing the number of output bits we can use.

The main contribution of this paper is showing how transforming the NFSR can reduce the attack complexity by reducing the cardinality of the dependence set of the output function.

Suppose that the output function can be expressed as $z(x) = x_a \oplus x_{a+i} \oplus z'(x)$, such that $a \notin dep(z')$, $a + i \notin dep(z')$ and $i > 1$. We want to eliminate $x_a$ from the output function. We note that if we shift the product term $p = x_{a+1}$ from $f_{a+i}$ to $f_{a+i-1}$, we get a new output function $\bar{z}(x) = z(T_{x_{a+1},a+i}(x)) = x_a \oplus (x_{a+i} \oplus x_a) \oplus z'(x) = x_{a+i} \oplus z'(x)$. Note that because $a + i \notin dep(z')$, the transform does not modify $z'$.

While this approach reduces the cardinality of the output function, it may complicate the feedback functions. In particular, for the case of Fibonacci NF-

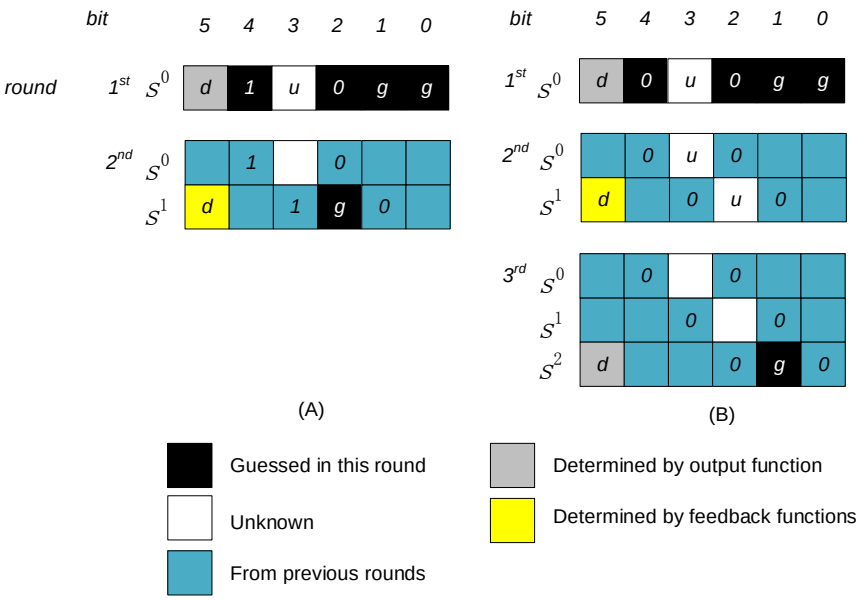


Fig. 5: Improved attack on NFSR $N_1$. (A) when $s_2^0 = 0$ and $s_3^1 = 1$. (B) when $s_2^0 = 0$ and $s_3^1 = 0$

SRs, such a transformation introduces two non-trivial feedback functions at $f_{a+i}$ and $f_{a+i-1}$ that reduce the amount of state propagation we can use. Hence, transforming the NFSR presents a balancing act between the complexity of the feedback functions and the cardinality of the dependence set of the output function.

Depending on the structure of the original NFSR, we can use two heuristics to reduce the effect of the transformation on the structure of the feedback functions. Assuming a Fibonacci NFSR, when $a+i$ is close to $n-1$, we use multiple shifts to shift $x_{n-i}$ from $f_{n-1}$ to $f_{a+i-1}$. such a shift will only result in one non-trivial feedback function at $f_{a+i-1}$. Furthermore, if we have $a < b < a+i$, such that

$$z(x) = x_a \oplus x_b \oplus x_{a+i} \oplus x_{b+i} \oplus z'(x) \tag{1}$$

and $a, b, a+i, b+i \notin dep(z')$, shifting $x_{n-i}$ from $f_{n-1}$ to $f_{a+i-1}$ would eliminate both $x_a$ and $x_b$ from the output function, while only introducing one non-trivial feedback function at $f_{a+i-1}$.

### 4.5 Attacking a Transformed NFSR

Observing $N_1$ we note that there is a combination of four product terms in the output function which satisfies the criteria outlined in Equation 1. Therefore, we shift the product term $x_2$ from the bit 5 to the bit 4, and then to the bit 3 to produce the following equivalent Galois NFSR $N_2$:

$$\begin{aligned} f_5(x) &= x_0 \oplus x_1 \oplus x_1x_2 \\ f_4(x) &= x_5 \\ f_3(x) &= x_4 \oplus x_0 \\ f_2(x) &= x_3 \\ f_1(x) &= x_2 \\ f_0(x) &= x_1 \end{aligned}$$

with a transformed output function we use is

$$z(x) = x_2x_3 \oplus x_4 \oplus x_5$$

Now we perform the guess-and-determine attack on $N_2$. In the first round we guess bits $s_2^0$, $s_3^0$ and $s_4^0$ and use the output function to determine $s_5^0$. We now propagate the known state we know the values of $s_1^1$, $s_2^1$ and $s_4^1$. However, because the feedback function $f_3$ is not trivial we do not know $s_3^1$, so we guess it, and can use the second output bit to determine $s_5^1$.

We now have guesses for $s_3^1$ and $s_4^0$ combining these with $s_3^1 = f_3(s^0) = s_4^0 \oplus s_0^0$ allows us to determine $s_0^0$.

If $s_2^0 = 0$, we can use $s_0^0$ with $s_5^1 = f_5(s^0) = s_0^0 \oplus s_1^0 \oplus s_1^0 s_2^0 = s_0^0 \oplus s_1^0 \oplus s_1^0 0 = s_0^0 \oplus s_1^0$ to determine $s_1^0$. This recovers the complete initial state $s^0$. This case is displayed in Figure 6 (A).

If $s_2^0 = 1$ (Figure 6 (B)), we can reduce $f_5(s^0) = s_0^0 \oplus s_1^0 \oplus s_1^0 s_2^0 = s_0^0 \oplus s_1^0 \oplus s_1^0 1 = s_0^0$. Because $s_0^0$ is determined via $f_3$, it is independent of $s_5^1$. Hence there is a 50% chance that the system is inconsistent, with $s_0^0 \neq s_5^1$. When the system is consistent, we propagate the data we have to $s^2$, guess $s_3^2$ and determine $s_5^2$ and we have the complete state.

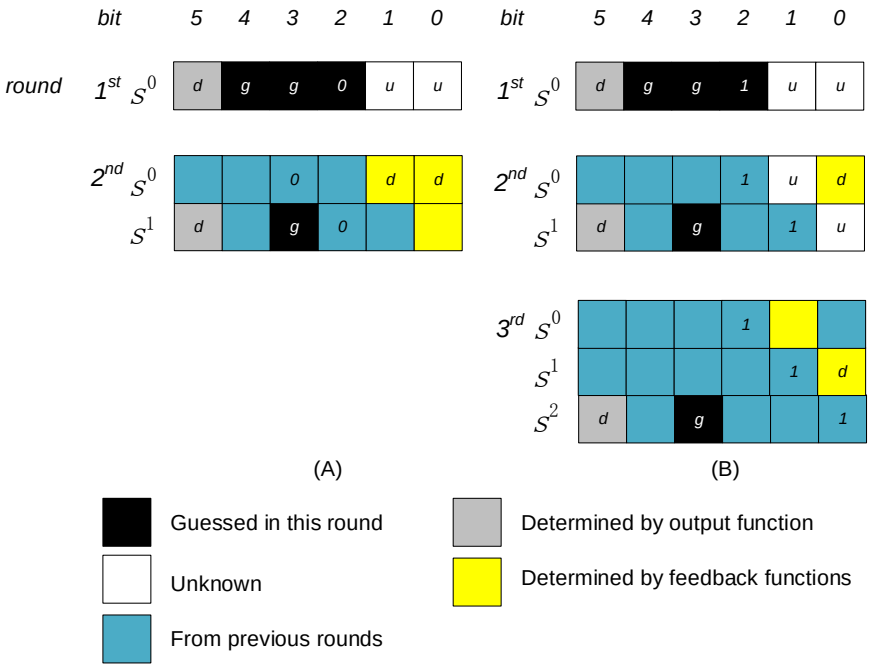


Fig. 6: Guess-and-determine attack on NFSR $N_2$. (A) $s_2^0 = 0$, (B) $s_2^0 = 1$

To sum up, we guess four bits for the first two rounds. In 3/4 of the cases we can completely determine the internal state. For the other 1/4 we need an additional guess. Hence, the attack complexity is $\frac{3}{4}2^4 + \frac{1}{4}2^5 = 20 = 2^{4.32}$. Admittedly, this is a very small improvement over the attack on the original NFSR. However, this is a toy example. As we shall now see, for larger NFSRs the improvement can be much more significant.

# 5 A Guess-and-Determine Attack on the Sprout Stream Cipher

So far we have demonstrated that NFSR transformations can improve guess-and-determine attacks only in a very limited scenario. We now show that the technique also works for real ciphers.

In this section we describe an attack on the Sprout cipher [3]. We start with a description of the cipher, proceed with a description of a transformation that reduces the cardinality of the output function and implement a guess-and-determine attack on the cipher.

## 5.1 Description of Sprout Stream Cipher

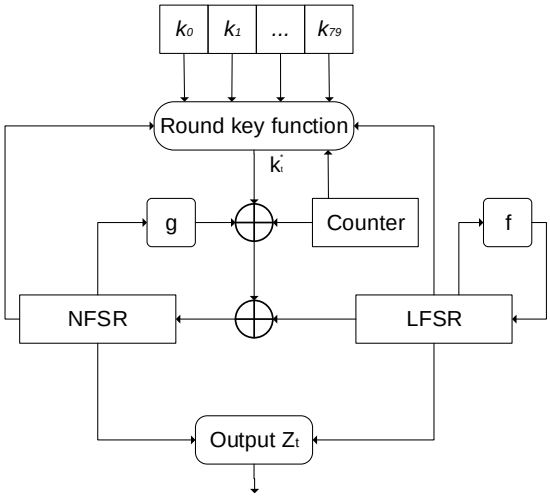


Fig. 7: Sprout cipher

Sprout [3] is a Grain-family stream cipher, which is inspired by Grain-128a [2]. Its internal state consists of a 40-bit LFSR $l$, a 40-bit Fibonacci NFSR $n$ and a 7-bit counter $c$. The counter selects key bits that are fed together with counter bit $c_4^t$ to the feedback function of the NFSR. Sprout uses an 80 bit key and a 70 bit *initialisation vector* (IV).

To describe the cipher We adopt a notations similar to that of Lallemand and Naya-Plasencia [22]:

– $t$: clock-cycle number.

- $l_i^t$ the $i^{\text{th}}$ state of the LFSR at clock $t$.
- $n_i^t$ the $i^{\text{th}}$ state of the NFSR at clock $t$.
- $k = (k_0, k_1, ..., k_{79})$ the key bits.
- $IV = (IV_0, IV_1, ..., IV_{69})$ initialisation vector.
- $k_t^t$ round key bit generated at clock $t$.
- $z_t$ keystream bit generated at clock $t$.
- $c_t$ the value of the counter at clock $t$.

The LFSR uses the following feedback function:

$$l_{39}^{t+1} = l_0^t \oplus l_5^t \oplus l_{15}^t \oplus l_{20}^t \oplus l_{25}^t \oplus l_{34}^t$$

The NFSR uses the following feedback function:

$$\begin{aligned} n_{39}^{t+1} =& k_t^* \oplus l_0^t \oplus c^t \oplus n_0^t \oplus n_{13}^t \oplus n_{19}^t \oplus n_{35}^t \oplus n_{39}^t \oplus n_2^t n_{25}^t \oplus n_3^t n_5^t \oplus n_7^t n_8^t \oplus n_{14}^t n_{21}^t \\ & \oplus n_{16}^t n_{18}^t \oplus n_{22}^t n_{24}^t \oplus n_{26}^t n_{32}^t \oplus n_{33}^t n_{36}^t n_{37}^t n_{38}^t \oplus n_{10}^t n_{11}^t n_{12}^t \oplus n_{27}^t n_{30}^t n_{31}^t \end{aligned}$$

where $k_t^*$ is defined as:

$$k_t^* = \begin{cases} k_t & 0 \leq t \leq 79 \\ (k_{t \bmod 80})(l_4^t \oplus l_{21}^t \oplus l_{37}^t \oplus n_9^t \oplus n_{20}^t \oplus n_{29}^t) & t \geq 80 \end{cases}$$

The output function is:

$$z_t = n_4^t l_6^t \oplus l_8^t l_{10}^t \oplus l_{32}^t l_{17}^t \oplus l_{19}^t l_{23}^t \oplus n_4^t n_{38}^t l_{32}^t \oplus l_{30}^t \oplus n_1^t \oplus n_6^t \oplus n_{15}^t \oplus n_{17}^t \oplus n_{23}^t \oplus n_{28}^t \oplus n_{34}^t$$

When the cipher starts, the NFSR and the LFSR are initialised from the IV using the following formulas: $n_i^0 = IV_i$, $0 \leq i \leq 39$, $l_i^0 = IV_{i+40}$, $0 \leq i \leq 29$, and $l_i^0 = 1$, $30 \leq i \leq 38$, $l_{39}^0 = 0$. The cipher then executes 320 initialisation rounds, in which rather than producing output, $z_t$ is fed back into the feedback functions of the LFSR and the NFSR to update internal state.

### 5.2 Transformation of Sprout Cipher

We observe that the four linear product terms $n_{17}, n_{23}, n_{28}, n_{34}$ of the output function meet the requirements of Equation 1, with $a = 17$, $b = 23$ and $i = 11$. We, therefore apply the transformation suggested above by shifting $p = x_{n-i} = x_{29}$ from $f_{39}$ to $f_{27}$.

We note that $x_{29}$ is not in the ANF of the feedback function. Consequently, moving it from the feedback function will *add* the product term, making the function slightly more complex. Furthermore, for each shift we execute, we might have to update the feedback function as described in Theorem 4. The resulting

feedback function is:

$$\begin{aligned} n_{39}^{t+1} =& k_t^* \oplus l_0^t \oplus c^t \oplus n_0^t \oplus n_{13}^t \oplus n_{17}^t \oplus n_{18}^t \oplus n_{19}^t \oplus n_{24}^t \oplus n_{28}^t \oplus n_{29}^t \oplus n_{35}^t \oplus n_{39}^t \\ & \oplus n_2^t n_{25}^t \oplus n_3^t n_5^t \oplus n_7^t n_8^t \oplus n_{14}^t n_{21}^t \oplus n_{16}^t n_{18}^t \oplus n_{22}^t n_{24}^t \oplus n_{26}^t (n_{32}^t \oplus n_{21}^t) \\ & \oplus (n_{33}^t \oplus n_{22}^t)(n_{36}^t \oplus n_{25}^t)(n_{37}^t \oplus n_{26}^t)(n_{38}^t \oplus n_{27}^t) \oplus n_{10}^t n_{11}^t n_{12}^t \\ & \oplus n_{27}^t (n_{30}^t \oplus n_{19}^t)(n_{31}^t \oplus n_{20}^t) \\ n_{27}^{t+1} =& n_{28}^t \oplus n_{17}^t \end{aligned}$$

We also note that $k_t^*$ depends on some internal state that is modified by the transform hence we need to update it to:

$$k_t^* = \begin{cases} k_t & 0 \le t \le 79 \\ (k_{t \bmod 80})(l_4^t \oplus l_{21}^t \oplus l_{37}^t \oplus n_9^t \oplus n_{18}^t \oplus n_{20}^t \oplus n_{29}^t) & t \ge 80 \end{cases}$$

Finally, the output function depends on $n_{38}$. Consequently, while the transformation removes $n_{17}$ and $n_{23}$ from the output function, it also adds a dependency on $n_{27}$. Thus the cardinality of the output function is only reduced by 1. The new output function is:

$$z_t = n_4^t l_6^t \oplus l_8^t l_{10}^t \oplus l_{32}^t l_{17}^t \oplus l_{19}^t l_{23}^t \oplus n_4^t l_{32}^t (n_{38}^t \oplus n_{27}^t) \oplus l_{30}^t \oplus n_1^t \oplus n_6^t \oplus n_{15}^t \oplus n_{28}^t \oplus n_{34}^t$$

For initialisation, we apply the shift state transformation of Definition 1 to initialise the 40-bit NFSR of the transformed Sprout by:

$$(IV_0, IV_1, ..., IV_{27}, IV_{28} \oplus IV_{16}, IV_{29} \oplus IV_{17}, ..., IV_{39} \oplus IV_{28})$$

### 5.3 Implementation of Guess-and-Determine Attack on the Sprout

In each of the first 8 rounds, we use one keystream bit to determine one bit in the internal state. In the first round, we guess 15 bits (8 from the LFSR and 7 from the NFSR), and determine one bit by the output function (see Figure 8. For simplicity, we do not present the status of the LFSR in the diagram.

In the second round, we again guess 15 bits and determine one (Figure 9). Additionally, because we know both $n_{27}^1$ and $n_{28}^0$ we can use the feedback function of $n_{27}^1$ to determine $n_{17}^0 = n_{16}^1$.

From the third round forward, the state propagation provides us with some guesses to the dependence set of the output function. Consequently, we only need to guess 9 bits (4 LFSR and 5 NFSR) in the third round. (Figure 10.) The number of bits we guess keeps decreasing—we only need to guess 8, 6, 6, 5 and 3 bits for the fourth, fifth, sixth, seventh and eighth rounds, respectively. These are shown in Figures 11–15. During these round, we continue to use the feedback function of $n_{27}$ to determine $n_{17}$.

For the ninth round we know all but four of the inputs of the output function, so we need to guess three to determine the fourth. We guess two additional state bits in the NFSR (see Figure 16) to increase the number of NFSR bits we know.

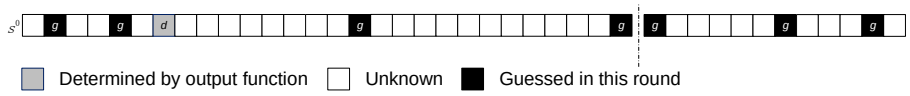


Fig. 8: The 1st round of guess-and-determine attack on Sprout

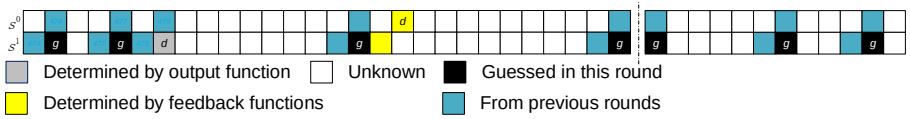


Fig. 9: The 2nd round of guess-and-determine attack on Sprout

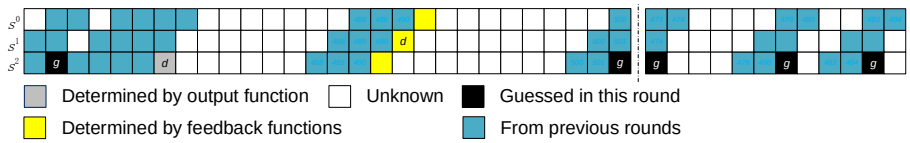


Fig. 10: The 3rd round of guess-and-determine attack on Sprout

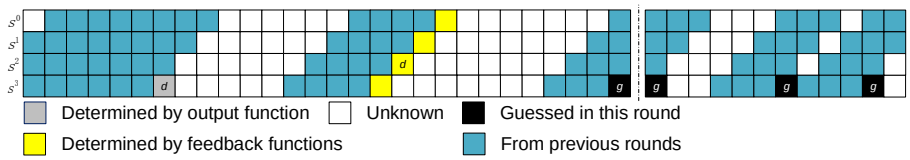


Fig. 11: The 4th round of guess-and-determine attack on Sprout

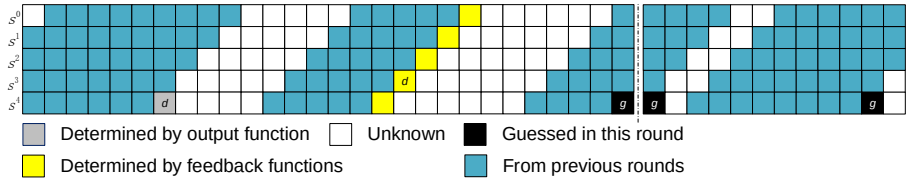


Fig. 12: The 5th round of guess-and-determine attack on Sprout

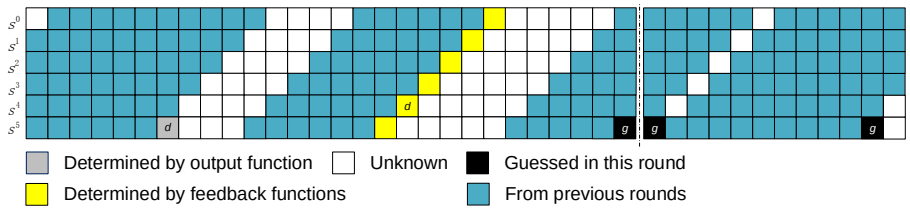


Fig. 13: The 6th round of guess-and-determine attack on Sprout

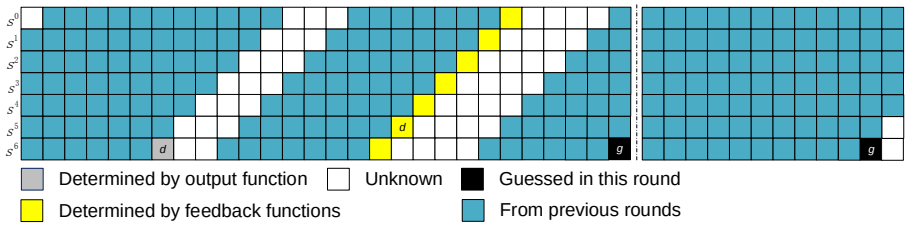


Fig. 14: The 7th round of guess-and-determine attack on Sprout

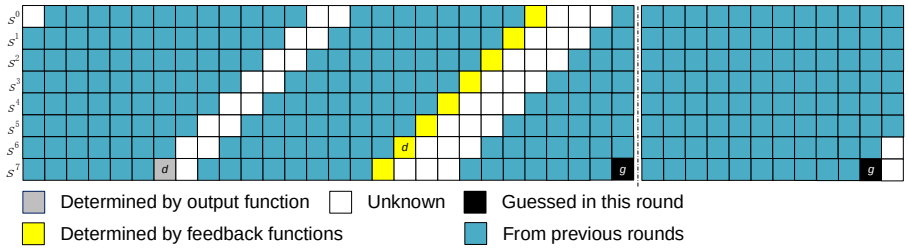


Fig. 15: The 8th round of guess-and-determine attack on Sprout

At this stage most of the state has been guessed and we can start exploiting the non-linear terms of the output function to determine some more bits and to reject inconsistent states. (See Section 4.3.) If we are able to determine all of the state bits, we are done. Otherwise we need to guess an additional bit and repeat the attempt to complete the state.

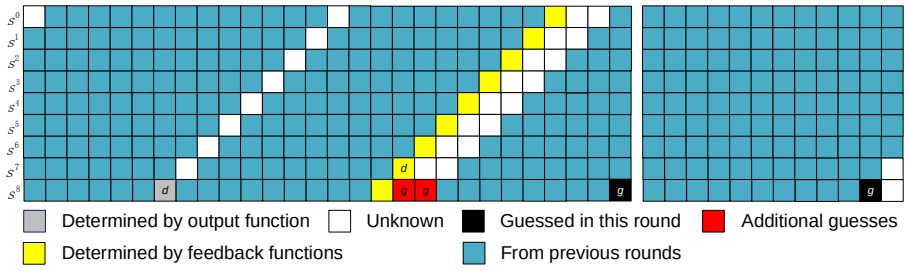


Fig. 16: The 9th round of guess-and-determine attack on Sprout

Once the internal state is recovered we follow the technique of Lallemand and Naya-Plasencia [22] to recover the key. Each round there is a 50% probability of having a key bit in the feedback function. Hence, by running the cipher for 160 rounds we expect to have 60 of the key bits affecting at least one of these

160 rounds. We need to run the cipher for four more rounds for the key bit to affect the output function. Hence after 164 rounds we can expect to recover 60 key bits. We can then brute force the remaining 20 bits.

To estimate the complexity of the attack, we ran 1,000,000 instances of the last 3 rounds and counted the number of guesses in each. More specifically, for each of the 1,000,000 instances we chose a random key, a random IV and a random guess of 59 bits guessed in for first six rounds of the attack. We then count the number of guesses we make for rounds 7 to 9. The average number of guesses we make is $3775.10 = 2^{11.87}$. For the first six rounds we need to guess a total of 59 bits. Hence, the attack complexity is $2^{59+11.87} = 2^{70.87}$ This compares favourably with the attacks of Lallemand and Naya-Plasencia [22] and Banik [5] which require $2^{74.51}$ and $2^{75}$ guesses, respectively. The attack is significantly slower than the TMD attack of Esgin and Kara [16], however, our attack has modest memory requirements whereas their attack requires over 770 terabytes of memory.

## 6 Conclusion

In this paper we show that transformations of NFSRs from the Fibonacci configuration to the Galois configuration can assist cryptanalysis. To demonstrate this, we generalise and extend past work on NFSR transformations. We discuss the conditions under which such transformations can facilitate cryptanalytic attacks and demonstrate the use of the technique to cryptanalyse the Sprout cipher. Our guess-and-determine attack is faster than all previously published non-TMD attacks on Sprout.

Our technique essentially shift complexity from the output function to the feedback functions of the NFSR. While we are able to demonstrate improvement in cryptanalysis, further work is required for better understanding the trade-offs in NFSR transformations. Such understanding is also likely to help in designing more secure NFSRs.

# Bibliography


[1] Mohamed Ahmed Abdelraheem, Julia Borghoff, Erik Zenner, and Mathieu David. Cryptanalysis of the light-weight cipher A2U2. In *Proceedings of the 13th IMA International Conference on Cryptography and Coding (IMACC)*, pages 375–390, Oxford, UK, December 2011.

[2] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of Grain-128 with optional authentication. *International Journal of Wireless and Mobile Computing*, 5(1):48–59, December 2011.

[3] Frederik Armknecht and Vasily Mikhalev. On lightweight stream ciphers with shorter internal states. In *Proceedings of the 22nd International Workshop on Fast Software Encryption (FSE)*, pages 451–470, Istanbul, Turkey, mar 2015.

[4] S. H. Babbage. Improved "exhaustive search" attacks on stream ciphers. In *European Convention on Security and Detection*, pages 161–166, Brighton, UK, May 1995.

[5] Subhadeep Banik. Some results on Sprout. Cryptology ePrint Archive Report 2015/327, April 2015.

[6] Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *Advances in Cryptology—Crypto 2006*, pages 1–21, Santa Barbara, CA, US, aug 2006.

[7] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In *Advances in Cryptology—ASIACRYPT 2000*, pages 1–13, Kyoto, Japan, December 2000.

[8] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In *Fast Software Encryption*, pages 1–18, New York, NY, US, April 2001.

[9] Daniel Bleichenbacher and Sarvar Patel. SOBER cryptanalysis. In *Fast Software Encryption*, pages 305–316, Rome, Italy, March 1999.

[10] Nicolas T. Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology—EUROCRYPT 2003*, pages 345–359, Warsaw, Poland, May 2003.

[11] Elena Dubrova. A transformation from the Fibonacci to the Galois NLFSRs. *IEEE Transactions on Information Theory*, 55(11):5263–5271, November 2009.

[12] Elena Dubrova. Finding matching initial states for equivalent NLFSRs in the Fibonacci and the Galois configurations. *IEEE Transactions on Information Theory*, 56(6):2961–2966, June 2010.

[13] Elena Dubrova. A list of maximum period NLFSRs. Cryptology ePrint Archive Report 2012/166, March 2012.

[14] Elena Dubrova. An equivalence-preserving transformation of shift registers. In *Sequences and Their Applications*, pages 187–199, Melbourne, VIC, Australia, November 2014.

[15] Patrik Ekdahl and Thomas Johansson. SNOW-a new stream cipher. In *First Open NESSIE Workshop*, pages 167–168, Leuven, Belgium, November 2000.

[16] Muhammed F. Esgin and Orhun Kara. Practical cryptanalysis of full Sprout with TMD tradeoff attacks. Cryptology ePrint Archive Report 2015/327, September 2015.

[17] Xiutao Feng, Jun Liu, Zhaocun Zhou, Chuankun Wu, and Dengguo Feng. A byte-based guess and determine attack on SOSEMANUK. In *Advances in Cryptology—ASIACRYPT 2010*, pages 146–157, Singapore, December 2010.

[18] Jovan Dj Golić. Cryptanalysis of alleged A5 stream cipher. In *Advances in Cryptology—EUROCRYPT*, pages 239–255, Konstanz, Germany, May 1997.

[19] Philip Hawkes and Gregory G. Rose. Exploiting multiples of the connection polynomial in word-oriented stream ciphers. In *Advances in Cryptology—ASIACRYPT 2000*, pages 303–316, Kyoto, Japan, December 2000.

[20] Alexander Klimov and Adi Shamir. A new class of invertible mappings. In *4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002)*, pages 470–483, Redwood Shores, CA, US, August 2002.

[21] Lars R. Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen, and Sven Verdoolaege. Analysis methods for (alleged) RC4. In *Advances in Cryptology—ASIACRYPT'98*, pages 327–341, Beijing, China, October 1998.

[22] Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of full sprout. In *Advances in Cryptology—Crypto 2015*, Santa Barbara, CA, US, Oct 2015.

[23] Zhiqiang Lin. The transformation from the Galois NLFSR to the Fibonacci configuration. In *Emerging Intelligent Data and Web Technologies (EIDWT)*, pages 335–339, Xi'an, China, September 2013.

[24] Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989.

[25] Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic attacks and decomposition of Boolean functions. In *Advances in Cryptology—EUROCRYPT 2004*, pages 474–491, Interlaken, Switzerland, May 2004.

[26] María Naya-Plasencia. How to improve rebound attacks. In *Advances in Cryptology–CRYPTO 2011*, pages 188–205, Santa Barbara, CA, US, August 2011.

[27] Zhenqing Shi, Xiutao Feng, Dengguo Feng, and Chuankun Wu. A real-time key recovery attack on the lightweight stream cipher A2U2. In *Cryptology and Network Security—CANS 2012*, pages 12–22, Darmstadt, Germany, December 2012.

[28] Hongjun Wu. *Cryptanalysis and Design of Stream Ciphers.* PhD thesis, Katholieke Universiteit Leuven, July 2008.

[29] Yuval Yarom, Gefei Li, and Damith C. Ranasinghe. Evaluation and cryptanalysis of the Pandaka lightweight cipher. In *13th International Conference on Applied Cryptography and Network Security*, New York, NY, US, June 2015.

[30] Bin Zhang and Dengguo Feng. New guess-and-determine attack on the self-shrinking generator. In *Advances in Cryptology—ASIACRYPT 2006*, pages 54–68, Shanghai, China, December 2006.