To Shift or Not to Shift: Understanding GEA-1

Christof Beierle[1], Patrick Felke[2] and Gregor Leander[1]

[1] Ruhr University Bochum, Bochum, Germany firstname.lastname@rub.de
[2] University of Applied Sciences, Emden/Leer, Germany patrick.felke@hs-emden-leer.de

Abstract. In their Eurocrypt 2021 paper, Beierle et al. showed that the proprietary stream ciphers GEA-1 and GEA-2, widely used for GPRS encryption in the late 1990s and during the 2000s, are cryptographically weak and presented attacks on both algorithms with practical time complexity. Although GEA-1 and GEA-2 are classical stream ciphers, the attack on GEA-1 is interesting from a cryptanalytic point of view. As outlined in the aforementioned paper, there is a strong indication that the security of GEA-1 was deliberately weakened to 40 bits in order to fulfill European export restrictions. In this paper we analyze the design further and answer the open question on how to construct a GEA-1-like cipher with such a reduced security. Indeed, the actual GEA-1 instance could be obtained from this construction. Our observations and analysis yields new theoretical insights in designing secure stream ciphers.

Keywords: cryptanalysis · GPRS · GEA-1 · stream cipher · LFSR · 40-bit security

1 Introduction

General Packet Radio Service (GPRS) is a mobile data standard based on the GSM (2G) technology, widely deployed during the late 1990s and the early 2000s. At the cryptographic level, the data processed by the GPRS protocol is protected by a stream cipher and, initially, the proprietary encryption algorithm GEA-1 was designed by ETSI Security Algorithms Group of Experts (SAGE) in 1998 (and the successor algorithm GEA-2 one year later) for this purpose. To generate the keystream, GEA-1 gets as input a 64-bit key, a 32-bit IV, and a 1-bit flag to indicate the direction of communication. A technical report on the design process can be found at [ETS98].

GEA-1 was designed at a time in which export restrictions were still in place. While the key length was 64 bit, in [BDL+21] it was shown that the effective key length is below, i.e., there exists a practical attack of a complexity of 2^{40} GEA-1 evaluations. There is no definite answer whether or not this weakness is a result of the Wassenaar arrangement. Although some of our analysis is in this direction, we will never be able to give a proof.

The full specification of the stream ciphers GEA-1 and GEA-2 can be found in [BDL+21]. More precisely, GEA-1 consists of linear feedback shift registers (LFSRs) that are regularly clocked. It is worth mentioning that those LFSRs are actually implemented in Galois mode, i.e., instead of summing the bits at several tap positions to compute the new state, a single bit is added to several bit positions. It is well-known that LFSRs in Galois mode are equally good as LFSRs in Fibbonacci mode from a cryptoanalytic point of view. They are the preferred choice when a fast implementation in software is required (see [Sch96]). Therefore it is worth mentioning that GEA-1 and GEA-2 are both implemented in hardware. A non-linear and cryptographically strong Boolean function is used on each register by extracting a few bits at fixed position and producing one bit per register and clock cycle. The output keystream bit of GEA-1 is then simply the sum, i.e., the XOR, of those three bits. In other words, GEA-1 can be seen as the sum of three filtered LFSRs. GEA-2 is very

similar in design and (besides small and seemingly minor changes in the initialization) uses a fourth LFSR in order to cope with the larger key and thus larger state.

In [BDL+21] the authors showed that the two algorithms do not achieve an adequate security level. Indeed, they presented attacks on `GEA-1` and `GEA-2` with complexities corresponding to a security of 40 bits and approximately 45 bits respectively. While the attack on `GEA-2` is a combination of several techniques, most of which have been developed within the last two decades, the attack on `GEA-1` is based on a simple but very remarkable observation. Indeed, after the linear initialization procedure the joint state of two of the involved LFSRs have a joint entropy of only 40 bits, while their size adds up to 64 bits. That is, the initialization is far from being optimal and indeed 24 bits of information are disregarded in this process. This loss of entropy directly leads to a classical meet-in-the-middle attack: guess the joint state of the two LFSRs and compare the keystream with the help of a table for the third LFSR.

The authors of [BDL+21] further analyze how frequently this surprising observation occurs for randomly chosen LFSRs. For this, they replaced the (two) LFSRs used in `GEA-1` by randomly chosen primitive LFSRs in Galois mode of the corresponding size and computed the loss of entropy. After roughly one million tries, the maximal loss that was observed was at most 9 bits,[1] demonstrating that this behavior is (i) very rare and thus (ii) most likely built in to keep the ciphers effective strength at 40 bits.

One important question was not answered in [BDL+21], namely: How was this configuration constructed? By extrapolating the experimental observations given in [BDL+21, Table 2], we estimate the cost of constructing this simply by randomly picking primitive LFSRs to be in the range of roughly 2^{47} tries, summing up to around 2^{65} binary operations in total.[2] Taking into account that the design is already more than 20 years old, the cost of this would have been enormous. This strongly indicated that there must be a more elaborated and efficient way of achieving the desired setting.

1.1 Our Contribution

In this paper, we give a very efficient way how to construct LFSRs such that the number of possible joint states is much less than expected, thus explaining how the weakness could have been, and assumingly has been, constructed. The main objective of interest here is the intersection of the kernels of the linear initialization process.

For this we describe the states, the initialization and the intersection of states by polynomials over $\mathbb{F}_2$. In a nutshell a key t is in the kernel of the first LFSR, if its polynomial representation is a multiple of the characteristic polynomial of the LFSR. For the second LFSR, things are slightly different due what to the fact that the key is shifted before the initialization. This leads to the fact the same key t is in the kernel of the second LFSR if its *rotated version* is a multiple of the characteristic polynomial of the second LFSR. This immediately implies that not using a shift would make such a weakness impossible, as the polynomial corresponding to the key simply cannot be a multiple of two different primitive polynomials due to its natural degree restriction. Thus, this a priori small effect of shifting the key before initialization turns out to be the key-enabler to control the actual security in the first place. This is astonishing as one would have expected otherwise at the first glance, i.e. one would have expected that shifting increases the security. While this explains that the shift is needed to get a high dimensional intersection of kernels, it still does not give rise to a construction of the LFSRs themselves. This requires two more ingredients. First, as we will detail in Section 2, considering keys with

[1] When considering all possible combinations of two of the three registers in `GEA-1`, the maximal observed loss was 11 bits.

[2] We estimate the number of expected solutions to be $s \cdot 2^{-2d+1}$, where s denotes the sample space and d the desired entropy loss. For each sample, we need to solve a linear system of dimension 64 to compute the entropy loss.

a number of zero bits at well-chosen positions lead to multiple, linear independent, keys in the intersection automatically, as long as at least one key in the intersection exists. Second, we show that reversing the design process allows to get the needed boost in success probability. Instead of picking LFSRs, we start by picking an element in the kernel by factorizing the corresponding polynomial description. If the polynomial by chance has a primitive divisor of the required degree, we can simply choose this divisor as the primitive polynomial defining the LFSR. In other words we pre-describe the sought for kernel to some extend and build the LFSRs around it.

We show by decomposing the kernel of `GEA-1` that it can be easily constructed by our approach. As `GEA-1` is then only one example of the (re?)-discovered design strategy, we elaborate in Section 4.2 about other possible parameter choices and also about limits for this approach.

As a side remark, the above mentioned use of LFSRs in Galois mode can now also be justified: An Fibonacci LFSR that is based on a random characteristic polynomial, and thus very likely with many taps, is very unfavorable to implement in software and thus very unusual. For a LFSR in Galois mode, the choice of a random characteristic polynomial with many taps is desirable.

2 Preliminaries

We briefly introduce the mathematics behind GEA-like stream ciphers. As usual a matrix $A \in \mathbb{F}_2^{m\times n}$ is considered as linear mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ via matrix-vector multiplication from the right. Thus the i-th column of A is the image of $e_i := \underbrace{(0,0,\ldots,1,0\ldots,0)^\top}_{i-\text{th position}} \in \mathbb{F}_2^n$, the i-th canonical unit vector and A is the representation matrix with respect to this canonical basis.

2.1 The Galois Mode

First we recall some basic facts about linear feedback shift registers (LFSRs) in Galois mode, as depicted in Figure 1. For further reading we refer to ([Sch96, p. 378 ff.],[HK04, p. 227]).

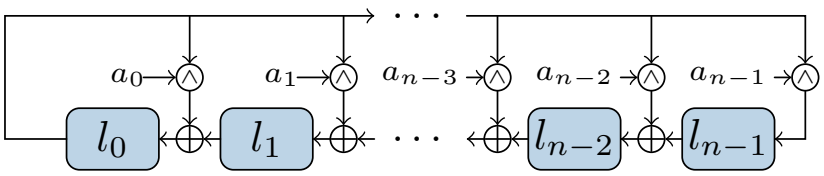

Figure 1: An LFSR in Galois mode.

Given an LFSR L in Galois mode of degree n with entries in $\mathbb{F}_2$, clocking the inner state $l = l_0, \ldots, l_{n-1}$ is equivalent to the matrix-vector multiplication

$$G_L \cdot l := \begin{pmatrix} a_0 & 1 & 0 & \ldots & 0 \\ a_1 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-2} & 0 & 0 & \ldots & 1 \\ a_{n-1} & 0 & 0 & \ldots & 0 \end{pmatrix} \cdot \begin{pmatrix} l_0 \\ l_1 \\ \vdots \\ l_{n-2} \\ l_{n-1} \end{pmatrix} = \begin{pmatrix} a_0 l_0 + l_1 \\ a_1 l_0 + l_2 \\ \vdots \\ a_{n-2} l_0 + l_{n-1} \\ a_{n-1} l_0 \end{pmatrix}.$$

The characteristic polynomial of G_L is

$$g(X) := X^n + a_0 X^{n-1} + \cdots + a_{n-2} X + a_{n-1}\,.$$

Although LFSR's in Galois mode can be defined more general we only consider the cases where $g(X)$ is primitive as only those are of cryptographic interest. Thus the characteristic polynomial $g(X)$ is equal to the minimal polynomial of G_L and $a_{n-1} \neq 0$. The latter condition is equivalent to the fact that G_L is invertible. Vice versa, given a primitive polynomial $g(X) \coloneqq X^n + a_0X^{n-1} + \cdots + a_{n-2}X + a_{n-1}$ then

$$G_L \coloneqq \begin{pmatrix} a_0 & 1 & 0 & \dots & 0 \\ a_1 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-2} & 0 & 0 & \dots & 1 \\ a_{n-1} & 0 & 0 & \dots & 0 \end{pmatrix}$$

is the (invertible) companion matrix of an LFSR in Galois mode with minimal polynomial g. This can be seen as follows. Let

$$S \coloneqq \begin{pmatrix} 0 & 0 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \iddots & \vdots \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \end{pmatrix} .$$

Then $S^2 = I_n$, where I_n denotes the $n \times n$ identity matrix and

$$S^{-1}G_LS = SG_LS = \begin{pmatrix} 0 & 0 & \dots & 0 & a_{n-1} \\ 1 & 0 & \dots & 0 & a_{n-2} \\ 0 & 1 & \dots & 0 & a_{n-3} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \vdots & 1 & a_0 \end{pmatrix} , \tag{1}$$

which is the well-known Frobenius companion matrix with characteristic polynomial (and minimal polynomial) $g(X)$ (see [LN96], chapter 2.5). We call such a matrix the *Galois matrix* of degree n and the corresponding minimal polynomial the *Galois polynomial* in the sequel. If the degree n is obvious we will omit it. Moreover, given an LFSR L in Galois mode with minimal polynomial g, we also denote the Galois matrix with G_g if appropriate. As usual $G_g^0 \coloneqq I_n$. The reason why these kind of registers are called LFRS in Galois Mode will become obvious in Section 4. Moreover usually the matrix given in equation (1) is the preferred choice when working with LFSRs in Galois mode. Our representation is a direct consequence of the actual implementation of `GEA-1`,`GEA-2` (see [BDL+21]) and thus better suited for our paper.

3 `GEA-1` and its Cryptoanalytic Properties

In this section we recall the description of the stream cipher `GEA-1` as well as its weakness, both as presented in [BDL+21].

3.1 Description of `GEA-1`

Keystream generation. The keystream is generated from three LFSRs over $\mathbb{F}_2$, called A, B and C, together with a 7-bit non-linear filter function f. The registers A, B, C have lengths $31, 32$ and 33, respectively and the LFSRs work in Galois mode. In particular, the

Galois polynomials corresponding to LFSRs A, B, C are

$$\begin{aligned} g_A(X) &= X^{31} + X^{30} + X^{28} + X^{27} + X^{23} + X^{22} + X^{21} + X^{19} + X^{18} + X^{15} \\ &\quad + X^{11} + X^{10} + X^8 + X^7 + X^6 + X^4 + X^3 + X^2 + 1\,, \\ g_B(X) &= X^{32} + X^{31} + X^{29} + X^{25} + X^{19} + X^{18} + X^{17} + X^{16} + X^9 + X^8 \\ &\quad + X^7 + X^3 + X^2 + X + 1\,, \\ g_C(X) &= X^{33} + X^{30} + X^{27} + X^{23} + X^{21} + X^{20} + X^{19} + X^{18} + X^{17} + X^{15} \\ &\quad + X^{14} + X^{11} + X^{10} + X^9 + X^4 + X^2 + 1\,, \end{aligned}$$

respectively. The specification of $f = f(x_0, x_1, \ldots, x_6)$ is given in algebraic normal form as

$$\begin{aligned} &x_0x_2x_5x_6 + x_0x_3x_5x_6 + x_0x_1x_5x_6 + x_1x_2x_5x_6 + x_0x_2x_3x_6 + x_1x_3x_4x_6 \\ &+x_1x_3x_5x_6 + x_0x_2x_4 + x_0x_2x_3 + x_0x_1x_3 + x_0x_2x_6 + x_0x_1x_4 + x_0x_1x_6 \\ &+x_1x_2x_6 + x_2x_5x_6 + x_0x_3x_5 + x_1x_4x_6 + x_1x_2x_5 + x_0x_3 + x_0x_5 + x_1x_3 \\ &+x_1x_5 + x_1x_6 + x_0x_2 + x_1 + x_2x_3 + x_2x_5 + x_2x_6 + x_4x_5 + x_5x_6 + x_2 + x_3 + x_5\,, \end{aligned}$$

and belongs to a class of cryptographically strong Boolean functions that can be decomposed into two bent functions on 6 bits. For the considerations below, the choice of f is irrelevant, the constructions focuses on the choice of the LFSRs only.

When all registers have been initialized (see below), the actual keystream generation starts. This is done by taking the bits in seven specified positions in each register to be the input to f. The outputs from the three f-functions are XORed together to produce one bit of the keystream. Figure 2 shows the particular feedback positions of each register, as well as showing which positions form which input to f. In Figure 2, the topmost arrow in the input to f represents x_0, and the input at the bottom is x_6. After calculating the keystream bit, all registers are clocked once each before the process repeats.

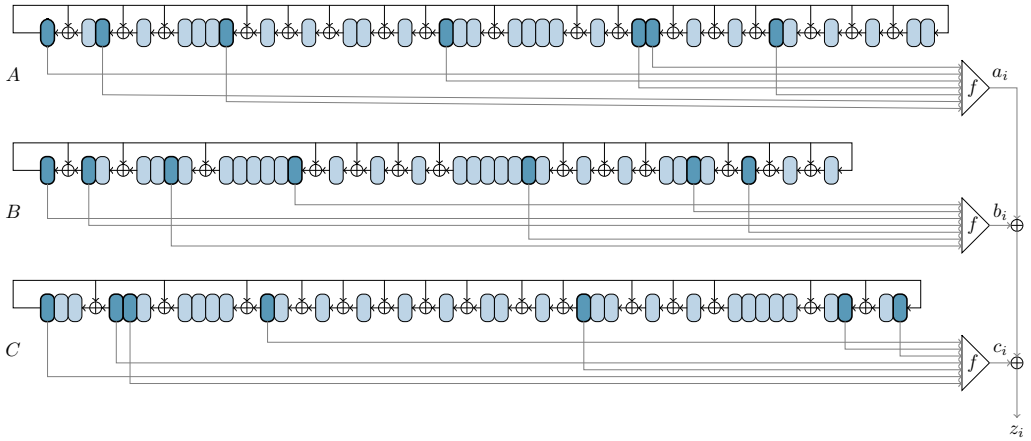

Figure 2: Overview of the keystream generation of `GEA-1` [BDL+21].

Initialization. The cipher is initialized via a non-linear feedback shift register S of length 64. This register is filled with zeros at the start of the initialization process. The input for initializing `GEA-1` consists of a public 32-bit initialization vector IV, one public bit *dir* (indicating direction of communication/uplink or downlink in a cellular network), and a 64-bit secret key K. The initialization starts by clocking S for 97 times, feeding in one input bit with every clock. The input bits are introduced in the sequence

$IV_0, IV_1, \ldots, IV_{31}, dir, K_0, K_1, \ldots, K_{63}$. When all input bits have been loaded, the register is clocked another 128 times with zeros as input. The feedback function consists of f, XORed with the bit that is shifted out and XORed with the next bit from the input sequence. Figure 3 depicts the particular tap positions.

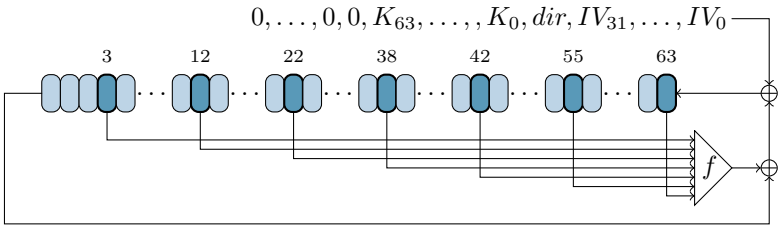

Figure 3: Initialization of register S [BDL+21].

After S has been clocked 225 times, the content of the register is taken as a 64-bit string $s = s_0, \ldots, s_{63}$. This string is taken as a seed for initializing A, B and C as follows. First, all three registers are initialized with zeros. Then each register is clocked 64 times, with an s_i-bit XORed onto the bit that is shifted out before feedback. Register A inserts the bits from s in the natural order $s_0, s_1, ..., s_{63}$. The sequence s is cyclically shifted by 16 positions before being inserted to register B, so the bits are entered in the order $s_{16}, s_{17}, \ldots, s_{63}, s_0, \ldots, s_{15}$. For register C the sequence s is cyclically shifted by 32 positions before insertion starts. Figure 4 depicts the process for register B. Note that if any of the registers A, B or C end up in the all-zero state, the bit in position 0 of the register is forcibly set to 1 before keystream generation starts.

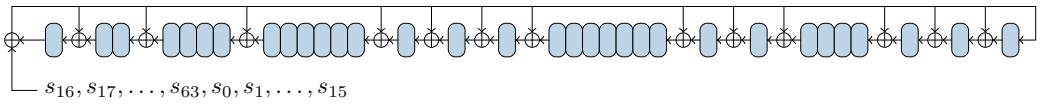

Figure 4: Initialization of register B [BDL+21].

As already observed in [BDL+21], if we exclude the unlikely case that any of the three registers A, B or C is still in the all-zero state after the shifted insertion of s, the initialization process of the three registers with the string s is obviously linear and therefore there exist three matrices $M_A \in \mathbb{F}_2^{31\times 64}$, $M_B \in \mathbb{F}_2^{32\times 64}$ and $M_C \in \mathbb{F}_2^{33\times 64}$ such that $\alpha = M_A s, \beta = M_B s$, and $\gamma = M_C s$, where α, β and γ denote the states of the three LFSRs after the initialization phase.

3.2 The Attack on GEA-1

Let us consider the initialization matrices $M_A \in \mathbb{F}_2^{31\times 64}$, $M_B \in \mathbb{F}_2^{32\times 64}$ and $M_C \in \mathbb{F}_2^{33\times 64}$ such that

$$\begin{aligned} \alpha &= M_A s \,, \\ \beta &= M_B s \,, \\ \gamma &= M_C s \,. \end{aligned}$$

We exclude here the unlikely case that α, β or γ is still in the all-zero state after the shifted insertion of s. Theses three matrices have full rank. This implies that the number of possible starting states after initialization is maximal when each LFSR is considered independently, i.e. there are 2^{31} possible states for register A, 2^{32} possible states for register B, and 2^{33} possible states for register C, as should be expected. This corresponds

to the linear mappings represented by M_A, M_B and M_C having kernels of dimension of at least 33, 32 and 31, respectively. However, when considering pairs of registers, one gets a decomposition of $\mathbb{F}_2^{64}$ as a direct sum of the kernels of the linear mappings. In [BDL+21] it was observed that if one denotes $T_{A,C} \coloneqq \ker(M_A) \cap \ker(M_C)$ and $U_B \coloneqq \ker(M_B)$, one gets

1. $\dim(T_{A,C}) = 24$,
2. $\dim(U_B) = 32$,
3. $U_B \cap T_{A,C} = \{0\}$.

From this it directly follows that $\mathbb{F}_2^{64}$ can be decomposed into the direct sum $U_B \oplus T_{A,C} \oplus V$, where V is of dimension 8. Thus, for the key-dependent and secret string s, there exists a *unique* representation $s = u + t + v$ with $u \in U_B$, $t \in T_{A,C}$, $v \in V$ and

$$\begin{aligned}\beta &= M_B(u + t + v) = M_B(t + v)\\ \alpha &= M_A(u + t + v) = M_A(u + v)\\ \gamma &= M_C(u + t + v) = M_C(u + v)\ .\end{aligned}$$

Indeed, from this decomposition, s can be computed with a meet-in-the-middle attack with a complexity[3] of 2^{37} `GEA-1` evaluations to build (and sort) a table with 2^{32} entries of size 97 bit (65 bit of keystream to reconstruct the key uniquely with high probability and 32 bit for v,t leading to this keystream) and a brute-force step of complexity 2^{40} `GEA-1` evaluations for each new session key $K_0, \ldots, K_{63}$. For more details of the attack see [BDL+21]. Note that once s is recovered it is easy to recover $K_0, \ldots, K_{63}$ by clocking the S-register backwards. Hence, the attack has to be conducted only *once per GPRS session* and is done in 2^{40} operations once the table has been established. In other words, the joint state of A and C can be described with only 40 bits and thus can take only 2^{40} possible values. This is the key observation of the attack and in [BDL+21] it is claimed that such a decomposition of the key space is highly unlikely to occur accidentally, as it is argued by computer simulations. The main question arising in this context is how to design such a system. As demonstrated by the experiments conducted in [BDL+21] a trial and error approach is elusive. This question will be answered in the next section.

4 Shifting Matters

In this section we will give a method to build ciphers of `GEA-1` type which are vulnerable to the attack described above and thereby answering the corresponding question in [BDL+21].

It will become apparent without giving a rigorous proof that systems of `GEA-1` type having a keyspace that can be decomposed into a direct sum similar as above only appear for *very special choices* of the shift constants together with *very special choices* for the Galois polynomials. This is astonishing because one could intuitively expect that shifting following valid well-known principles, as done in `GEA-1`, only strengthens the system.

In general, this demonstrates that it is not recommended to modify the canonical way of feeding the key into the LFSRs. Indeed, only by modifying this initialization and by using the shifted key for the individual LFSRs, the attack becomes possible.

To describe our construction, we need the following properties of Galois matrices. These are well-known facts from classical ring or field theory respectively (e.g., see [LN96, War94]). To keep the paper self-contained and enhance readability we added a proof.

[3] The complexity will be measured by the amount of operations that are roughly as complex as `GEA-1` evaluations (for generating a keystream of size ≤ 128 bit).

Theorem 1. *Given a Galois matrix G_g of degree n with primitive Galois polynomial g. Let $\mathbb{F}_2[G_g] := \{\sum_{i=0}^{m} t_i G_g^i \mid m \in \mathbb{N}, t_i \in \mathbb{F}_2, i = 0, \ldots, m\}$ be the subring generated by all linear combinations of powers of G_g in $\mathbb{F}_2^{n\times n}$ and $(g(X))$ denote the ideal generated by $g(X)$ in $\mathbb{F}_2[X]$. Then the following statements are true.*

1. *The mapping*
$$\begin{aligned}\psi: \quad & \mathbb{F}_2[X]/(g(X)) \longrightarrow \mathbb{F}_2[G_g] \\ & \textstyle\sum_{i=0}^{m} t_i X^i \mapsto \sum_{i=0}^{m} t_i G_g^i\end{aligned}$$
is a ring isomorphism. Thereby $\sum_{i=0}^{m} t_i X^i$ denotes a representative of an element of the quotient ring.

2. *Every element $v \in \mathbb{F}_2^n, v \neq 0$ is G_g-cyclic, i.e. the set $\{G_g^i v : i = 0, \ldots, n-1\}$ forms a basis of $\mathbb{F}_2^n$.*

Proof. It is a well-known fact from algebra that for an irreducible polynomial $q(X)$ the set of all polynomials with $p(M) = 0$ for a matrix M with minimal polynomial $q(X)$ is equal to the ideal $(q(X))$ and $\mathbb{F}_2[X]/(q(X))$ is a finite field of degree n. By our general agreement $g(X)$ is primitive, therefore irreducible and the minimal polynomial of G_g. Thus the canonical mapping $\phi : \mathbb{F}_2[X] \longrightarrow \mathbb{F}_2[G_g], p(X) \mapsto p(G_g)$ has kernel $(g(X))$ and is therefore an isomorphism by the well-known isomorphism theorem for quotient rings. For a fixed $v \in \mathbb{F}_2^n, v \neq 0$, consider $\sum_{i=0}^{n-1} t_i G_g^i v = \left(\sum_{i=0}^{n-1} t_i G_g^i\right) v = 0$, where not all t_i are equal to zero. By applying the isomorphism ψ^{-1} we get that $\sum_{i=0}^{n-1} t_i X^i \neq 0 \in \mathbb{F}_2[X]/(g(X))$ as the degree of the polynomial $\sum_{i=0}^{n-1} t_i X^i$ is at most $n-1$ and not all t_i are equal to zero. As $\mathbb{F}_2[X]/(g(X))$ is a finite field, the element $\sum_{i=0}^{n-1} t_i X^i \in \mathbb{F}_2[X]/(g(X))$ is invertible. Let $\sum_{i=0}^{n-1} t'_i X^i$ denote its inverse. Then by applying ψ it follows that $\sum_{i=0}^{n-1} t_i G_g^i$ is invertible with inverse $\sum_{i=0}^{n-1} t'_i G_g^i$. Hence $\left(\sum_{i=0}^{n-1} t_i G_g^i\right) \cdot v \neq 0$ as $v \neq 0$. This is a contradiction and therefore, v is G_g-cyclic. □

Remark 1. Note that $\mathbb{F}_2[X]/(g(X))$ is a field. The matrix $G'_g := S^{-1} G_g S$ (see Section 2.1, equation (1)) is the representation matrix of the linear mapping $P(X) \mapsto XP(X)$ over the finite field $\mathbb{F}_2[X]/(g(X))$ with respect to the basis $1, \ldots, X^{n-1}$, i.e. the i-th column yields $XX^i = \sum_{j=1}^{n} G'_{g\,j,i} X^{j-1} \in \mathbb{F}_2[X]/(g(X))$. This connection to a central mapping, the so-called left multiplication, in the theory of finite fields, which form an important class of Galois fields, lead to the name for these kind of LFSR. Note that G_g is the representation matrix of the left multiplication with respect to the above basis in reverse order, i.e. $X^{n-1}, \ldots, 1$.

The following corollary is extensively used in the remainder of this paper. The proof is a straightforward application of Theorem 1 and therefore omitted.

Corollary 1. *Let e_0 denote the vector $e_0 := (1, 0, \ldots, 0)^\top \in \mathbb{F}_2^n$ and G_g a Galois matrix of degree n for a primitive polynomial g. Then, for $m \geq 0, t_i \in \mathbb{F}_2, i = 0, \ldots, m$, we have $\sum_{i=0}^{m} t_i G_g^i e_0 = 0$ if and only if $g(X)$ divides $\sum_{i=0}^{m} t_i X^i$ in $\mathbb{F}_2[X]$.*

4.1 The Impact of Shifting

At first we will settle the question how to find two primitive Galois polynomials g_1 and g_2 of degree d_1 and d_2, respectively, such that for the corresponding Galois matrices G_{g_1}, G_{g_2}, the dimension $\dim\left(T_{G_{g_1}, G_{g_2,cs}}\right)$ is at least ξ. Thereby $0 \leq cs \leq \kappa - 1$ denotes a cyclic shift employed during the initialization and κ the size of the session key (64 in case of `GEA-1`). Without loss of generality we focus on this case. It is a routine matter to extend the approach presented in the sequel to the case where in both initialization phases a shift is applied. First of all, note that the columns of $M_{G_{g_1}}$ (the initialization matrix without

a shift) consist of $G_{g_1}^{\kappa-i}e_0$, where $e_0 := (1,0,\dots,0)^\top \in \mathbb{F}_2^{d_1}$ and $i = 0,\dots,\kappa-1$ as the columns of $M_{G_{g_1}}$ consist of the images of $s_i := \underbrace{(0,0,\dots,1,0\dots,0)^\top}_{i-\text{th position}} \in \mathbb{F}_2^\kappa$. Note that we have to clock the register i times before the first bit of s_i not equal to zero is plugged into the register and thus the state becomes non-zero. This explains the shape of $M_{G_{g_1}}$. By Corollary 1 we have that

$$\sum_{i=0}^{\kappa-1} t_i G_{g_1}^{\kappa-i} e_0 = 0$$

if and only if $g_1(X)$ divides

$$\sum_{i=0}^{\kappa-1} t_i X^{\kappa-i} .$$

In the case of g_2, we get by similar reasoning and taking into account the effect of cs that the columns of $M_{G_{g_2},cs}$ (the initialization matrix with a shift) consist of

$$\begin{aligned} G_{g_2}^{cs-i}e_0 &\quad \text{if } i < cs \\ G_{g_2}^{\kappa-i+cs}e_0 &\quad \text{otherwise} , \end{aligned}$$

where $e_0 := (1,0,\dots,0)^\top \in \mathbb{F}_2^{d_2}$. Note that now we have to clock the register $j = \kappa + i - cs$ times before the first bit of s_i not equal to zero is plugged into the register and thus the state is non-zero. This gives the first case. The second follows in the same way. In the same vein as above we get that

$$\sum_{i=0}^{cs-1} t_i G_{g_2}^{cs-i} e_0 + \sum_{i=cs}^{\kappa-1} t_i G_{g_2}^{\kappa-i+cs} e_0 = 0$$

if and only if $g_2(X)$ divides

$$\sum_{i=0}^{cs-1} t_i X^{cs-i} + \sum_{i=cs}^{\kappa-1} t_i X^{\kappa-i+cs} .$$

Hence, a vector $(t_0,\dots,t_{\kappa-1})^\top \in \mathbb{F}_2^\kappa$ lies in $T_{G_{g_1},G_{g_2,cs}}$ if and only if $g_1(X)$ divides $\sum_{i=0}^{\kappa-1} t_i X^{\kappa-i}$ and $g_2(X)$ divides $\sum_{i=0}^{cs-1} t_i X^{cs-i} + \sum_{i=cs}^{\kappa-1} t_i X^{\kappa-i+cs}$. From this we get the following theorem which shows how to control the dimension of $T_{G_{g_1},G_{g_2,cs}}$.

Theorem 2. *Let g_1, g_2 be two primitive Galois polynomials and $0 \le cs \le \kappa-1$ be an integer. For $t = (t_0,\dots,t_{\kappa-1})^\top \in T_{G_{g_1},G_{g_2,cs}}$ we define the associated polynomials*

$$p_1(X) := \sum_{i=0}^{\kappa-1} t_i X^{\kappa-i}$$

and

$$p_2(X) := \sum_{i=0}^{cs-1} t_i X^{cs-i} + \sum_{i=cs}^{\kappa-1} t_i X^{\kappa-i+cs} .$$

Let

$$\begin{aligned} r_1 &= \min\{k : X^k \textit{ is a monomial with non-zero coefficient in } p_1\}, \\ r_2 &= \min\{k : X^k \textit{ is a monomial with non-zero coefficient in } p_2\}, \\ r_3 &= \min\{r_1, r_2\} \end{aligned}$$

Then

1. *for* $0 \leq s \leq r_3 - 1$, *the shifts* $t^s := (\underbrace{0, \ldots, 0}_{\times s}, t_0, t_1, \ldots, t_{\kappa-1-s})^\top$ *are elements of* $T_{G_{g1}, G_{g2,cs}}$.
2. *the elements* t^s *are linearly independent and thus span a subspace of dimension* r_3.

Proof. By definition, it is $t_{\kappa-1-(r_1-1)} = 1$, as it is the coefficient of X^{r_1} in p_1. We further have $t_{\kappa-1-(r_1-1)+1} = 0, t_{\kappa-1-(r_1-1)+2} = 0, \ldots, t_{\kappa-1} = 0$ from the definition of r_1. Hence the elements t^s are linearly independent.

As $t = (t_0, \ldots, t_{\kappa-1}) \in T_{G_{g_1}, G_{g_2,cs}}$ we have that $g_1(X)$ divides $p_1(X)$ and $g_2(X)$ divides $p_2(X)$. By definition of r_3, the associated polynomials of t^s are of the form $X^{-s}p_1(X), X^{-s}p_2(X)$ and still contained in $\mathbb{F}_2[X]$. Therefore they are divided by $g_1(X)$ and $g_2(X)$. Thus the elements t^s form a subspace of dimension r_3 of $T_{G_{g_1}, G_{g_2,cs}}$. □

4.2 Constructing The Galois Polynomials

The principle to construct systems vulnerable to the attack described in Section 3.2 is now fairly simple: We start with an element in the (potential) kernel, that would imply the desired dimension by the above theorem. We than construct the two polynomials p_1 and p_2 and check if they are divisible by primitive polynomials of the desired degree. We explain this in more detail below, first for the parameters used in `GEA-1` and second for the general case.

The Case of `GEA-1`. We give an example for the case of $\kappa = 64, cs = 32, \xi = 24$, and primitive polynomials g_1, g_2 of degree $d_1 = 31, d_2 = 33$. Those parameters correspond exactly to the case of `GEA-1`. Other parameter choices are discussed below.

First of all we will construct an element $t = (t_0, \ldots, t_{63})^\top$ of the form such that applying Theorem 2 yields $(t_0, \ldots, t_{63})^\top \in T_{G_{g_1}, G_{g_2,cs}}$, where $T_{G_{g_1}, G_{g_2,cs}}$ is of dimension 24. For this, let us fix t such that $t_i = 0$ for $i \in \{9, 10, \ldots, 31\} \cup \{41, \ldots, 63\}$ and $t_0 = t_{40} = 1$. We consider the polynomials

$$p_1 \coloneqq X^{64} + \sum_{i=1}^{8} t_i X^{64-i} + \sum_{i=32}^{39} t_i X^{64-i} + X^{24} \in \mathbb{F}_2[X]$$

$$p_2 \coloneqq X^{32} + \sum_{i=1}^{8} t_i X^{32-i} + \sum_{i=32}^{39} t_i X^{64-i+32} + X^{56} \in \mathbb{F}_2[X]$$

to guarantee a kernel of dimension at least 24 if there exist proper g_1, g_2 of degree 31,33 such that $t \in T_{G_{g_1}, G_{g_2,cs}}$. In the positive case the lower bound for the dimension (here 24) is a direct consequence of Theorem 2. We have 2^{16} possibilities to fix such an element t, i.e., to choose the above pair of polynomials p_1, p_2. We choose such an element t uniformly at random and check if p_1 is divided by a primitive polynomial g_1 of degree 31 and if p_2 is divided by a primitive polynomial g_2 of degree 33.

It is well known that the number of primitive elements of a finite field of q^n elements, where q is a prime number, is $\phi(q^n - 1)$ (see e.g., [Kob98, p. 56]). Here, ϕ denotes Euler's totient function. Hence the number of primitive polynomials in our case is $\frac{\phi(2^{31}-1)}{31}$ and $\frac{\phi(2^{33}-1)}{33}$, because the 31 (resp., 33) roots of a primitive polynomial of degree 31 (resp., 33) are all primitive. By construction $p_1 = X^{24} \cdot h_1(X)$, where $\deg(h_1(X)) = 40$, independently of the choice of the t_i. Analogously $p_2 = X^{24} \cdot h_2(X)$, where $\deg(h_2(X)) \leq 40$. The overall number of polynomials of degree 40 having a primitive divisor of degree 31 is $\frac{\phi(2^{31}-1)}{31} \cdot 2^9$ and $\frac{\phi(2^{33}-1)}{33} \cdot 2^8$ for polynomials of degree less or equal to 40. Therefore, under an independence assumption, we expect the probability that both h_1 has a primitive divisor of degree 31 and h_2 has a primitive divisor of degree 33 to be $\left(\frac{\phi(2^{31}-1)}{31 \cdot 2^{31}}\right)\left(\frac{\phi(2^{33}-1)}{33 \cdot 2^{33}}\right) \approx \frac{1}{1250}$.

As we have 2^{16} possibilities to vary p_1 and p_2, we expect to be successful to find the sought for polynomials g_1, g_2 with $t \in T_{G_{g_1},G_{g_2,cs}}$.

Indeed, the primitive polynomials g_A and g_C used in `GEA-1` are exactly of this form. More precisely, the element $t = (t_0, \ldots, t_{63})^\top$ with $(t_0, \ldots, t_8) = (1,0,1,1,0,0,1,1,1)$, $(t_{32}, \ldots, t_{40}) = (0,0,1,1,1,1,0,0,1)$, $t_i = 0$ for $i \in \{9, 10, \ldots, 31\} \cup \{41, \ldots, 63\}$ is contained in $T_{A,C}$. The corresponding polynomial p_1 is divided by

$$\begin{aligned} g_A(X) = {} & X^{31} + X^{30} + X^{28} + X^{27} + X^{23} + X^{22} + X^{21} + X^{19} + X^{18} + X^{15} \\ & + X^{11} + X^{10} + X^8 + X^7 + X^6 + X^4 + X^3 + X^2 + 1 \end{aligned}$$

and the corresponding polynomial p_2 is divided by

$$\begin{aligned} g_C(X) = {} & X^{33} + X^{30} + X^{27} + X^{23} + X^{21} + X^{20} + X^{19} + X^{18} + X^{17} + X^{15} \\ & + X^{14} + X^{11} + X^{10} + X^9 + X^4 + X^2 + 1 \,. \end{aligned}$$

As expected by Theorem 2, the 24-dimensional linear space $T_{A,C}$ is spanned by the shifted elements $t^s = (\underbrace{0, \ldots, 0}_{\times s}, t_0, t_1, \ldots, t_{63-s})^\top$, $0 \le s \le 23$.

From the 2^{16} possibilities to choose t, except from the example given above, also 47 other choices yield primitive divisors of p_1 and p_2 with degree 31 and 33, respectively. Note that once we have have been successful in finding the primitive polynomials g_1 and g_2, we could choose a primitive polynomial g_3 of degree 32 and check if $U_{G_{g_3}} \cap T_{G_{g_1},G_{g_2,cs}} = \{0\}$ in order to construct a stream cipher similar to `GEA-1`. In Appendix A, we provide a sage [Sag21] code that allows to construct such weak instances of `GEA-1` based on this construction.

We have seen by the algorithm above that it is possible to find a shift cs and corresponding polynomials such that the resulting system can be broken with the attack described in Section 2.1.

The General Case. We now focus on the case of an arbitrary (even) key length κ and are aiming at constructing two LFSRs of size ℓ_1 and ℓ_2 such that the kernel has a dimension of (at least) ξ. In order to simplify the discussion and the notation, we focus on the case of $cs = \kappa/2$. The case of other shift values can be handled in a similar way as long as $cs \ge \xi$.

In order to construct the two LFSRs, i.e., the corresponding primitive polynomials of degree ℓ_1 and ℓ_2, we start again by an element in the kernel that, due to Theorem 2, guarantees a kernel intersection of dimension at least ξ. That is, we consider a vector

$$t = (t_0, \ldots, t_{\kappa-1})^\top$$

such that

$$t_i = 0 \text{ for } i \in \left\{ \frac{\kappa}{2} - \xi + 1, \ldots, \frac{\kappa}{2} - 1 \right\} \cup \{\kappa - \xi + 1, \ldots, \kappa - 1\}$$

and $t_0 = t_{\kappa-\xi} = 1$. To this choice of t, we get the corresponding polynomials

$$p_1 \coloneqq X^\kappa + \sum_{i=1}^{\frac{\kappa}{2}-\xi} t_i X^{\kappa-i} + \sum_{i=\frac{\kappa}{2}}^{\kappa-\xi-1} t_i X^{\kappa-i} + X^\xi \in \mathbb{F}_2[X] \tag{2}$$

$$p_2 \coloneqq X^{\frac{\kappa}{2}} + \sum_{i=1}^{\frac{\kappa}{2}-\xi} t_i X^{\frac{\kappa}{2}-i} + \sum_{i=\frac{\kappa}{2}}^{\kappa-\xi-1} t_i X^{\kappa-i+\frac{\kappa}{2}} + X^{\xi+\frac{\kappa}{2}} \in \mathbb{F}_2[X] \,. \tag{3}$$

The number of vectors t and thus the number of pairs of polynomials (p_1, p_2) we can construct this way is

$$N = 2^{\kappa-2\xi} \,.$$

To successfully construct the LFSRs of dimension at least ξ, we require that p_1 is divisible by a primitive polynomial of degree ℓ_1 and p_2 is divisible by a primitive polynomial of degree ℓ_2. To analyze the successes probability, we use as before a heuristic approach. More precisely, we assume that (p_1, p_2) behaves as a uniformly and independently chosen pair of polynomials (of degree κ and less or equal to κ respectively) with respect to their probability of being divisible by primitive polynomials of the desired degree.

The number of polynomials of degree n with a primitive divisor of degree ℓ is, analogously as above, given by

$$P_{n,\ell} \coloneqq 2^{(n-\ell)} \frac{\phi(2^\ell - 1)}{\ell} .$$

In turn, the probability of a polynomial of degree n to be divisible by a primitive divisor of degree ℓ is

$$\Psi_\ell \coloneqq \frac{P_{n,\ell}}{2^n} = \frac{\phi(2^\ell - 1)}{\ell 2^\ell} .$$

Note that Ψ_ℓ is also the probability of a polynomial of degree *less or equal to* n to be divisible by a primitive divisor of degree ℓ, as both nominator and denominator are multiplied by a factor of two in this case.

While computing lower bounds on Euler's totient function is non trivial, for our purpose it is sufficient, easier, and more precise to compute $\phi(2^\ell - 1)$ for practical relevant values of ℓ. For $\ell \leq 512$ we computed explicitly that

$$\frac{2^\ell}{\phi(2^\ell - 1)} \leq 3.4 , \tag{4}$$

using a computer program.

Following the above heuristic on the random behavior of p_1 and p_2 we get that the probability for a successful construction for one fixed t is given by

$$\Psi_{\ell_1}\Psi_{\ell_2} = \left(\frac{\phi(2^{\ell_1} - 1)}{\ell_1 \cdot 2^{\ell_1}}\right)\left(\frac{\phi(2^{\ell_2} - 1)}{\ell_2 \cdot 2^{\ell_2}}\right).$$

From Equation (4), the expected number of tries until suitable polynomials are found can be bounded by

$$(\Psi_{\ell_1}\Psi_{\ell_2})^{-1} \leq 12\ell_1\ell_2$$

for $\ell_i \leq 512, i \in \{1, 2\}$. This shows that the approach is easily feasible for all practical relevant choices of ℓ_1 and ℓ_2 and can be expected to find valid solutions as long as the number of candidates N is larger than the expected number of tries. Note that for concrete parameters with a large ξ, ℓ_1, ℓ_2 it is better to check if $(\Psi_{\ell_1}\Psi_{\ell_2})^{-1} \leq N$ as N becomes relatively small and $12\ell_1\ell_2$ significantly larger than $(\Psi_{\ell_1}\Psi_{\ell_2})^{-1}$.

The Reciprocal Solution. Note that slightly more can be said about the structure of good choices for g_q and g_2 and the corresponding space $T_{G_{g_1},G_{g_2,cs}}$. For example looking at the reciprocal polynomials of g_1 and g_2 results in the same dimension for the kernel.

For this let κ be even and $cs = \frac{\kappa}{2}$. If $t = (t_0, t_1, \ldots, t_{\kappa-1})^\top \in T_{G_{g_1},G_{g_2,cs}}$ for two primitive polynomials g_1, g_2, we have $t^* = (t_{\kappa-1}, \ldots, t_1, t_0) \in T_{G_{g_1^*},G_{g_2^*,cs}}$, where $g_i^*(X) \coloneqq X^{\deg g_i} g_i(X^{-1})$ denotes the *reciprocal polynomial* of g_i. This can be seen as follows.

For the two polynomials $p_i, i \in \{1, 2\}$ as given in Theorem 2, we define $\tilde{p}_i(X) \coloneqq X^\kappa p_i(X^{-1})$. Then,

$$X\tilde{p_1}(X) = \sum_{i=0}^{\kappa-1} t_i^* X^{\kappa-i}$$

$$X\tilde{p_2}(X) = \sum_{i=0}^{cs-1} t_i^* X^{cs-i} + \sum_{i=cs}^{\kappa-1} t_i^* X^{\kappa-i+cs}$$

and $t^* \in T_{G_{g_1^*}, G_{g_2^*, cs}}$ if $g_i^*(X)$ divides $X\tilde{p}_i(X)$ for $i \in \{1, 2\}$. Since $\tilde{p}_i(X) = X^{\kappa - \deg p_i} p_i^*(X)$, this happens if $g_i^*(X)$ divides $p_i^*(X)$ for $i \in \{1, 2\}$. By assumption, $t \in T_{G_{g_1}, G_{g_2, cs}}$, so $g_i(X)$ divides $p_i(X)$ for $i \in \{1, 2\}$, and therefore $g_i^*(X)$ also divides $p_i^*(X)$ for $i \in \{1, 2\}$. Moreover $g_i(X)$ is primitive if and only if $g_i(X)^*$ is primitive.

Applicability to longer Keys. We recall that in the attack on `GEA-1`, the keyspace $\mathbb{F}_2^{64}$ was decomposed into the direct sum $U_B \oplus T_{A,C} \oplus V$ such that

$$\begin{aligned}\beta &= M_B(u + t + v) = M_B(t + v)\\ \alpha &= M_A(u + t + v) = M_A(u + v)\\ \gamma &= M_C(u + t + v) = M_C(u + v)\end{aligned}$$

where $\dim(U_B \oplus V) = 40$ and $\dim(T_{A,C} \oplus V) = 32$. In general, if the key size is κ, a straightforward divide-and-conquer attack could be applied by either building a table of size at least $2^{\dim(T_{A,C} \oplus V)}$ bitstrings and conducting exhaustive search of complexity $2^{\dim(U_B \oplus V)}$ cipher evaluations or vice versa.

Let us now discuss whether it is possible to build weak `GEA-1`-like instances operating on a larger keyspace. For $\kappa = 96$, it is possible to choose primitive polynomials g_A and g_C of degree 47 and 49, respectively, such that for the corresponding LFSRs A and C we have $\dim T_{A,C} = 44$ (where the shift for initializing LFSR C is $cs = 48$). Those parameters directly correspond to the maximal dimension that can be expected by the formulas above and are also verified experimentally (see below in Table 2). To find this specific polynomials we have checked if it is possible to have $\dim T_{A,C} \geq 42$ i.e. $\xi = 42$ and $\ell_1 = 47, \ell_2 = 49$. As $(\Psi_{\ell_1}\Psi_{\ell_2})^{-1} \approx 2500$ and $N = 2^{12} = 4096$ our approach should be successful with high probability. Indeed our algorithm computed the above solution with the even larger $T_{A,C}$ of dimension 44. Note that these parameters are chosen at the edge with respect to our theory. We could then choose a primitive polynomial g_B of degree 48 such that $\dim U_B = 48$ and such that the keyspace can be decomposed into $\mathbb{F}_2^{96} = U_B \oplus T_{A,C} \oplus V$ with $\dim V = 4$. Thus, we can break such a scheme with time complexity 2^{52} cipher evaluations and memory complexity $2^{48} \cdot 141$ bit.[4] The size of such a table is 4512 TiB.

For larger key sizes, this approach quickly gets infeasible. For example, if we would aim for a key length of $\kappa = 112$ bit (i.e., the minimum security required by NIST), we would choose g_A, g_B, and g_C of degrees 55, 56, and 57, respectively, such that $\dim T_{A,C} = 50$, $\dim U_B = 56$ and $\dim V = 6$. Other choices would only allow for other trade-offs between memory and computation, but not for reducing both. The divide-and-conquer (or meet-in-the-middle) attack against such a `GEA-1` instance would require

$$2^{\dim U_B} \cdot (\kappa + 1 + \dim T_{A,C} + \dim V) = 2^{56} \cdot (113 + 56) = 2^{56} \cdot 169$$

bit of memory, which corresponds to 1,384,448 TiB. Hence this approach is tailored to keys spaces of smaller sizes.

4.3 Experimental Verification

The part that remains to be done is to experimentally verify the plausibility of the heuristic. As explained below, the experiments nicely match the theoretical predictions.

Recall that we assume that the polynomials constructed for a vector $t \in \mathbb{F}_2^\kappa$ as given in Equation (2) and (3) behave as a uniform random pair of polynomials. In particular this

[4] The length of each entry in the table must be large enough to avoid false key candidates. Similarly as described in [BDL+21, Section 3.1], we assume that each bitstring in the table is of size $\ell + \dim(T_{A,C})$, where ℓ is the minimum integer such that $(1 - 2^{-\ell})^{2^\kappa} \geq 0.5$.

Table 1: For $\kappa = 64$ and $\xi \in \{23, 24, \ldots, 28\}$, this table shows the number of p_1 that yield primitive divisors of degree ℓ (first number in cell), the number of p_2 (with $cs = \frac{\kappa}{2} = 32$) that yield primitive divisors of degree ℓ (second number in cell), compared to the theoretical estimate $\Psi_\ell 2^{\kappa-2\xi}$ rounded to the closest integer (number in parentheses).

(ℓ, ξ)	23	24	25	26	27	28
29	8884 9178 (8988)	2190 2224 (2247)	442 544 (562)	76 114 (140)	24 40 (35)	0 0 (9)
30	4242 4322 (4351)	1132 1106 (1088)	274 270 (272)	72 44 (68)	16 10 (17)	0 0 (4)
31	9460 8720 (8456)	2158 2052 (2114)	506 518 (529)	126 106 (132)	44 26 (33)	18 14 (8)
32	3822 4108 (4096)	966 1110 (1024)	226 272 (256)	40 68 (64)	0 12 (16)	0 4 (4)
33	6416 6158 (6440)	1624 1584 (1610)	416 378 (402)	112 108 (101)	10 12 (25)	0 0 (6)
34	4900 5128 (5140)	1194 1268 (1285)	314 336 (321)	46 78 (80)	18 18 (20)	6 10 (5)

means that a fraction of Ψ_{ℓ_1} of the polynomials p_1 have a primitive divisor of degree ℓ_1 and, similarly, a fraction of Ψ_{ℓ_2} of the polynomials p_2 have a primitive divisor of degree ℓ_2.

For $\kappa = 64$ and values of ξ between 24 and 28, we computed the exact number of polynomials p_1 and p_2 and compared this to the theoretical estimate.

More importantly, we checked the behaviour of pairs directly. In order to limit the set of parameter to consider, we restricted to the case of $\ell_2 = \ell_1 + 2$ and $\kappa = \ell_1 + \ell_2$. For each $4 \leq \ell_1 \leq 67$ we compared the maximal dimension that could actually be constructed to the maximal dimension that could have been expected to be possible following the heuristic. The results, shown in Table 2 and verifiable by the sage code provided in Appendix A, suggest that the heuristic is plausible.

5 Conclusion

Feeding a session key into LFSRs by making use of shifts is common in many designs, e.g. besides in `GEA-1` and `GEA-2` it is also used in A5/1. Our work demonstrates that those shifts, together with a clever choice of feedback polynomials, can be used to deliberately weaken the construction.

We gave an explicit and efficient way to construct those choices for a large variety of parameters. For the exact parameters of `GEA-1` our construction includes the choices made for `GEA-1` indicating that this (or a related) strategy was used in the actual design process. On the positive side, we see again that, in line with [BDL+21] this is unlikely to happen unintentionally.

While our theory is described with a focus on LFSRs in Galois mode, it applies to LFSRs in Fibonacci mode as well. Consequently, we also checked if a similar attack is

Table 2: Experiments for parameters $\ell_2 = \ell_1 + 2, \kappa = \ell_1 + \ell_2$, for $4 \leq \ell_1 \leq 67$. The value $\bar{\xi}_{\max}$ gives the maximum ξ such that $\Psi_{\ell_1}\Psi_{\ell_2}2^{\kappa-2\xi} \geq 1$. The value $\xi_{\max}$ gives the maximum ξ for which there exist p_1, p_2 that yield primitive divisors g_1, g_2 of degree ℓ_1 and ℓ_2, respectively. The value # sol gives the number of such tuples (p_1, p_2). For all such solutions, the dimension of $T_{G_{g_1},G_{g_2},cs}$ is equal to $\xi_{\max}$.

ℓ_1	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ξ_{exp}	1	3	3	4	4	6	6	8	8	10	10	11	11	13	13	15
$\xi_{\max}$	3	2	3	4	4	5	6	7	7	11	9	11	10	13	13	15
# sol	2	8*	4	2	2	8	2	4	2	2	6	2	14	2	2	4
ℓ_1	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
ξ_{exp}	15	17	17	19	19	21	21	23	23	25	25	26	27	28	28	30
$\xi_{\max}$	16	16	18	20	20	21	22	23	22	23	24	26	27	28	28	29
# sol	2	8	2	2	2	4	4	2	2	14	10	4	2	6	2	6
ℓ_1	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
ξ_{exp}	30	32	32	34	34	36	36	38	38	40	40	42	42	44	44	46
$\xi_{\max}$	30	32	32	34	34	36	35	38	39	40	39	44	44	44	45	45
# sol	2	2	2	4	4	4	8	2	4	2	2	2	2	2	2	2
ℓ_1	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67
ξ_{exp}	46	48	48	50	50	52	52	54	54	55	56	57	58	59	60	61
$\xi_{\max}$	46	48	47	50	49	53	51	53	53	55	56	57	58	60	59	60
# sol	2	2	6	2	6	2	4	2	4	2	2	4	2	2	10	6

applicable to A5/1, but without success. This is probably not surprising as A5/1 makes use of primitive polynomials with a minimal amount of feedback taps and a stuttering mechanism. Even if a comparable decomposition of $\mathbb{F}_2^{64}$ would have existed for A5/1 it is unclear to us how to exploit it due to the stuttering LFSRs. Finally our answer to the question raised in the title is "if to shift, then double-check!"

Acknowledgments

This work was supported by the German Research Foundation (DFG) within the framework of the Excellence Strategy of the Federal Government and the States – EXC 2092 CaSa – 39078197.

References

[BDL+21] Christof Beierle, Patrick Derbez, Gregor Leander, Gaetan Leurent, Håvard Raddum, Yann Rotella, David Rupprecht, and Lukas Stennes. Cryptanalysis of the GPRS encryption algorithms GEA-1 and GEA-2. In Anne Canteaut and Francois-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021, Proceedings*, Lecture Notes in Computer Science. Springer, 2021.

[ETS98] ETSI. Security algorithms group of experts (sage); report on the specification, evaluation and usage of the gsm gprs encryption algorithm (gea). Technical Report. Available at https://www.etsi.org/deliver/etsi_tr/101300_101399/101375/01.01.01_60/tr_101375v010101p.pdf (accessed May 31, 2021), 1998.

[HK04] Kenneth Hoffman and Ray A. Kunze. *Linear Algebra*. PHI Learning, 2004.

[Kob98] Neal Koblitz. *Algebraic aspects of cryptography*, volume 3 of *Algorithms and computation in mathematics*. Springer, 1998.

[LN96] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2 edition, 1996.

[Sag21] Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.3)*, 2021. `https://www.sagemath.org`.

[Sch96] Bruce Schneier. *Applied cryptography - protocols, algorithms, and source code in C, 2nd Edition*. Wiley, 1996.

[War94] William P. Wardlaw. Matrix representation of finite fields. *Mathematics Magazine*, 67(4):289–293, 1994.

A Source Code to Generate GEA1-like Systems

Listing 1: weakgea.sage

```
#GEA-1 parameter
kappa = 64
xi = 24
l1 = 31
l2 = 33
l3 = 32

trys = 2**16

hkappa = int(kappa/2)
qkappa = int(kappa/4)

def getInitMatrix_fast(p,keyLength,shift):
  P.<x> = PolynomialRing(GF(2))
  l = p.degree()
  #construct transformation matrix A for LFSR in Galois mode
  A = companion_matrix(x^l+1)
  A[0] = list(p)[0:l][::-1]
  A = A.transpose()
  e0 = vector(GF(2),[1]+[0]*(l-1))
  M = zero_matrix(GF(2),l,keyLength)
  for c in range(keyLength):
    if (c < shift):
      M.set_column(c,A**(shift-c)*e0)
    else:
      M.set_column(c,A**(keyLength-c+shift)*e0)
  return M.transpose()

V = VectorSpace(GF(2),(kappa-2*xi));
Pol.<X> = PolynomialRing(GF(2));

success = False
ctr = 0
while (ctr < trys):
  v = V.random_element()
```

```
  p1 = X**kappa+X**xi
  j = 0
  for i in [1..(hkappa-xi)]+[hkappa..(kappa-xi-1)]:
    p1 = p1+v[j]*X**(kappa-i)
    j = j+1
  fp1 = list(p1.factor())

  for f in fp1:
    if ((f[0].degree()==l1) and f[0].is_primitive()):
      p2 = X**hkappa+X**(hkappa+xi)
      j = 0
      for i in [1..(hkappa-xi)]:
        p2 = p2+v[j]*X**(hkappa-i)
        j = j+1
      for i in [hkappa..(kappa-xi-1)]:
        p2 = p2+v[j]*X**(kappa-i+hkappa)
        j = j+1
      fp2 = list(p2.factor())

      for g in fp2:
        if ((g[0].degree()==l2) and g[0].is_primitive()):
          g1 = f[0]
          g2 = g[0]
          print('g1 = ', g1)
          print('g2 = ', g2)
          ctr = trys

Initmat2 = getInitMatrix_fast(g2,kappa,hkappa)
Initmat1 = getInitMatrix_fast(g1,kappa,0)

T = Initmat2.kernel().intersection(Initmat1.kernel())
print('dim T = ', T.dimension())

ctr = 0
while (ctr < trys):
  g3 = X**l3+1
  for i in [1..(l3-1)]:
    g3 = g3+(GF(2).random_element())*X**i
  if g3.is_primitive():
    Initmat3 = getInitMatrix_fast(g3,kappa,qkappa)
    if Initmat3.kernel().intersection(T).dimension()==0:
      print('g3 = ', g3)
      ctr = trys
    else:
      print('try again')
      ctr = ctr+1
```