Cryptanalysis of the GPRS Encryption Algorithms GEA-1 and GEA-2

Christof Beierle[1], Patrick Derbez[2], Gregor Leander[1], Gaëtan Leurent[3], Håvard Raddum[4], Yann Rotella[5], David Rupprecht[1], and Lukas Stennes[1]

[1] Ruhr University Bochum, Bochum, Germany
`firstname.lastname@rub.de`

[2] Univ Rennes, CNRS, IRISA, Rennes, France
`patrick.derbez@irisa.fr`

[3] Inria, Paris, France
`gaetan.leurent@inria.fr`

[4] Simula UiB, Bergen, Norway
`haavardr@simula.no`

[5] Université Paris-Saclay, UVSQ, CNRS, Laboratoire de Mathématiques de Versailles, Versailles, France.
`yann.rotella@uvsq.fr`

Abstract. This paper presents the first publicly available cryptanalytic attacks on the `GEA-1` and `GEA-2` algorithms. Instead of providing full 64-bit security, we show that the initial state of `GEA-1` can be recovered from as little as 65 bits of known keystream (with at least 24 bits coming from one frame) in time 2^{40} `GEA-1` evaluations and using 44.5 GiB of memory. The attack on `GEA-1` is based on an exceptional interaction of the deployed LFSRs and the key initialization, which is highly unlikely to occur by chance. This unusual pattern indicates that the weakness is intentionally hidden to limit the security level to 40 bit by design.
In contrast, for `GEA-2` we did not discover the same intentional weakness. However, using a combination of algebraic techniques and list merging algorithms we are still able to break `GEA-2` in time $2^{45.1}$ `GEA-2` evaluations. The main practical hurdle is the required knowledge of 1600 bytes of keystream.

Keywords: GPRS Encryption · Stream Cipher · Algebraic attacks · GEA.

1 Introduction

General Packet Radio Service (GPRS) is a mobile data standard based on the GSM (2G) technology. With its large deployments during the early 2000s worldwide, GPRS (including EDGE) was the technology for many of us, which provided us the first mobile Internet connection. While some countries are about to sunset 2G technology (or have already done so), other countries rely on GPRS as a fallback data connection. Consequently, the security of those connections *was and still is* relevant for a large user base. In the wireless medium, an attacker conducts an eavesdropping attack by merely sniffing the traffic in the victim's vicinity. To protect against eavesdropping GPRS between the phone and the base station, a stream cipher is used and initially two proprietary encryption algorithms `GEA-1` and `GEA-2` were specified.

Design Process of the GPRS Encryption Algorithm. A dedicated encryption algorithm for GPRS, now known as `GEA-1`, was designed by ETSI Security Algorithms Group of Experts (SAGE) in 1998. A technical report on the design process is available at [15]. The total budget spent was 429 man days and six organizations have been involved in the process. As seen in [15, Section 8], the following requirements were set for the design:

> The algorithm should be a stream cipher which gets a 64-bit key (Kc), a 32-bit IV, and a 1 bit flag to indicate the transfer direction as inputs and outputs a stream of 1,600 bytes.

It was explicitly mentioned as a design requirement that *"the algorithm should be generally exportable taking into account current export restrictions"* and that *"the strength should be optimized taking into account the above requirement"* [15, p. 10]. The report further contains a section on the evaluation of the design. In particular, it is mentioned that the evaluation team came to the conclusion that, *"in general the algorithm will be exportable under the current national export restrictions on cryptography applied in European countries"* and that *"within this operational context, the algorithm provides an adequate level of security against eavesdropping of GSM GPRS services"* [15, p. 13].

A successor algorithm, called `GEA-2`, was designed later. An official requirement specification by ETSI as for `GEA-1` is not publicly available. According to Charles Brookson in 2001, *"GEA2 was defined about a year later than GEA1 and was an improvement, which was allowed by the easing of export control legislation"* [8, p. 4].

The particular restrictions that `GEA-1` should fulfill in order to be exportable are not specified in the requirements.

Export Regulations. For a detailed survey on national and international regulations concerning the use, supply, import and export of cryptographic algorithms in the '90s, we refer to the *Crypto Law Survey* of Bert-Jaap Koops [23]. In France, rather strict regulations have been in place. In particular, until the late '90s, the use, supply, import and export of cryptography for providing confidentiality was subject to authorization by the prime minister. The requirements for obtaining such an authorization were not publicly available. To quote from [23], *"It was unclear to what extent the restrictive regulation was enforced in practice; it was rumoured to be widely ignored. It seemed impossible for individuals or enterprises to obtain authorisation for 'strong' cryptography. Even for state-owned industry,* *cryptography that does not serve military or high-grade security purposes had to be breakable. SCSSI, the office dealing with authorisation, rendered decisions without motivation.”*

In 1998, the French Decrees[1] 98-206 and 98-207 were announced, in which exceptions from such authorization or declaration have been defined. The three most interesting exceptions defined in Decree 98-206 with regard to our work can be translated as follows:

– Means and services of Cryptology for *“mobile phones for civil use that do not implement end-to-end encryption”* are exempt from authorization or declaration for supply, use, import and export.
– Means and services of Cryptology for *“Commercial civil base towers with the following characteristics: a) Limited to connection with cell phones that cannot apply cryptographic techniques to traffic between terminals, excepted on the direct link between cell phones and base stations b) And not allowing the use of cryptographic techniques to traffic excepted on the radio interface”* are exempt from authorization or declaration for supply, use and import (but not export).
– Means and services of Cryptology in which *“exhaustive search of all possible keys does not require more than* 2^{40} *trials with a simple test”* are exempt from authorization or declaration for use and import (but not for supply and export).

Interestingly enough, we will show later in Section 3 that `GEA-1` offers only 40-bit security.

1.1 Related Work and Reverse Engineering

In 2011, Nohl and Melette analyzed the security of GPRS traffic and showed that GPRS signals could easily be eavesdropped [29]. This was reported as a serious weakness, especially since some providers did not activate encryption at all. However, according to the authors, most operators at that time employed the proprietary encryption algorithms `GEA-1` or `GEA-2` for encrypting the GPRS traffic.

In the same talk, Nohl and Melette also reported the reverse-engineering of those encryption algorithms. Without presenting all of the specification details, the following properties of the design of `GEA-1` have been shown:

– It is a stream cipher which works on an internal state of 96 bits and uses a 64-bit key.
– A non-linear function is employed for initialization.[2]

[1] Available via `https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000000753702` and `https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000000753703`, accessed Oct-06, 2020.
[2] See minute 32:15 of the recorded talk.

– The state is kept in three registers of sizes 31, 32, and 33 bits.[3]
– The state update function is linear, i.e., the registers are LFSRs.
– The function that generates the output stream has algebraic degree 4.

The structure of the GEA-1 stream cipher as known from [29] is depicted in Figure 1. For GEA-2, it was reported that it employs a similar algebraic structure to its predecessor GEA-1. While the key size for GEA-2 is 64 bits as well, the internal state was reported to be of size 125 bits.

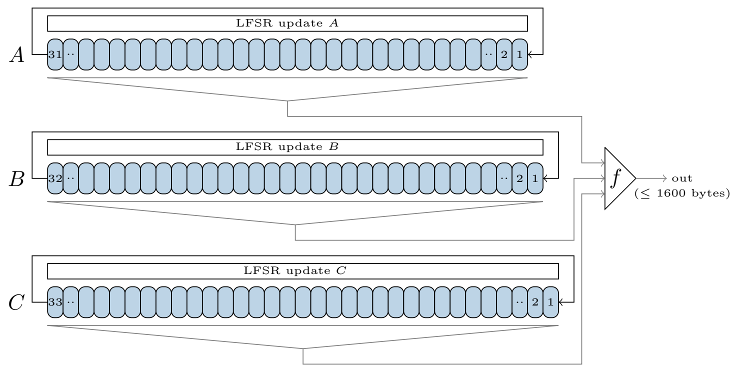

Fig. 1: The structure of the GEA-1 stream cipher with its 96 bit state known from [29]. The algebraic degree of the output function is 4.

In their talk, the authors claimed that GEA-1 has severe weaknesses against algebraic attacks, mainly due to the nonlinearity of the state update function and the availability of a long keystream to the adversary. Live on stage, a state-recovery attack was performed that took less than 15 minutes using "a Gaussian equation solver based on some SAT solver ideas" (minute 48:40 of the recorded talk).[4] However, details of this attack are not available.

Interestingly, the ETSI prohibited the implementation of GEA-1 in mobile phones in 2013, while GEA-2 and the non-encrypted mode are still mandatory to be implemented today [16].

Despite the hints of deliberately weakening GEA-1 for export and a demonstrated attack, a public cryptanalysis of GEA-1 and GEA-2 is still missing to date. This puts us in a position where we are uncertain about the algorithm's security guarantees. In this paper, we fill this gap with the first public cryptanalysis of GEA-1 and GEA-2. As part of this we also describe the design of those two proprietary algorithms, which we obtained from a source that prefers to stay anonymous.

[3] The size of the registers are visible in the live state-recovery attack, see minute 48:25 of the recorded talk.

[4] The authors acknowledged Mate Soos for ideas and also admitted that the live attack did not apply the SAT solver yet.

1.2 Our Contribution

After describing the stream ciphers `GEA-1` and `GEA-2` and their internal building blocks, we start by analyzing the security of `GEA-1`.

The main observation is that after the linear initialization process the *joint initial state of* 64 *bits* of two of the three LFSRs is guaranteed to be in one of only 2^{40} states (rather than close to 2^{64} as should be expected).

This property immediately allows to conduct a Divide-and-Conquer state-recovery attack in time 2^{40} `GEA-1` evaluations by using only 65 bits of known keystream (with at least 24 bits in the same frame). The attack needs the pre-computation of a (sorted) table of size 44.5 GiB, which can be done in time of roughly 2^{37} `GEA-1` evaluations. Once this table has been computed, the attack can be performed in time of roughly 2^{40} `GEA-1` evaluations for each new 64-bit session key.

Further, we experimentally show that for randomly chosen LFSRs, it is very unlikely that the above weakness occurs. Concretely, in a million tries we never even got close to such a weak instance. Figure 2 shows the distribution of the entropy loss when changing the feedback polynomials of registers A and C to random primitive polynomials. This implies that the weakness in `GEA-1` is unlikely to occur by chance, indicating that the security level of 40 bits is due to export regulations.

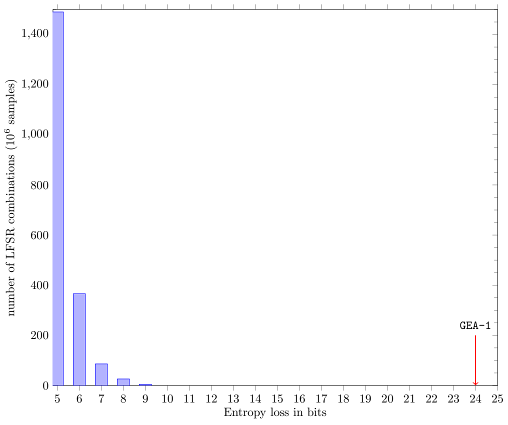

Fig. 2: The distribution of the entropy loss within the joint initial 64-bit state of registers A and C after the linear initialization in `GEA-1` for a random sample of 10^6 combinations of primitive LFSRs. The occurences of entropy losses up to 4 bits are omitted.

As a last part of this work, we look into the security of the successor algorithm `GEA-2`. We conduct a state-recovery attack that does not target the initialization process, but rather the keystream generation itself. The idea is to mix a list merging algorithm, combined with algebraic techniques. The attacks works in time equivalent to $2^{45.1}$ GEA-2 evaluations. The required memory is roughly 32 GiB. Rather than only 65 bit of known keystream as for GEA-1, this attacks needs all of the keystream available per frame, i.e., 1,600 bytes, and it cannot exploit information coming from multiple frames.

We demonstrate the practical feasibility of the attack against GEA-1 on standard hardware. Further, we discuss the real-world requirements and attack implications for today's mobile phone network. Eventually, we are dedicated to eliminating weak ciphers in current and future mobile phones — improving mobile network security.

2 Description of GEA-1 and GEA-2

In this section, we give a detailed description of the two algorithms GEA-1 and GEA-2, which we obtained from a source. Therefore we verify the correctness of the algorithms by *a*) using test vectors that are available on github [28] and *b*) verify the algorithm by checking the interoperability with commercial phones using the osmocom project [31]. Both experiments confirm the correct functionality; thus, we can assume that the provided algorithms are accurate with a high degree of certainty.

For the encryption, the GEA algorithms take the following input parameters: the plaintext, which is the GPRS LLC (Logical Link Control) frame, the key (K), the direction bit (uplink/downlink), and the IV (Input) that consists of an increasing counter for each frame.

As we will see, GEA-2 is an extension of GEA-1– with slight but crucial exceptions. For this reason, we first describe GEA-1 first and explain the differences and extensions for GEA-2 in a second step. An overview of the keystream generation of GEA-1 and GEA-2 is shown in Figure 3.

2.1 GEA-1

GEA-1 is built from three linear feedback shift registers over $\mathbb{F}_2$, called A, B and C, together with a non-linear filter function, called f. The registers A, B, C have lengths $31, 32$ and 33, respectively, and f is a Boolean function on seven variables of degree 4. The registers work in Galois mode. This means that if the bit that is shifted out of a register is 1, the bits in a specified set of positions in the register are flipped. The specification of $f = f(x_0, x_1, \ldots, x_6)$ is given in algebraic normal form as follows:

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

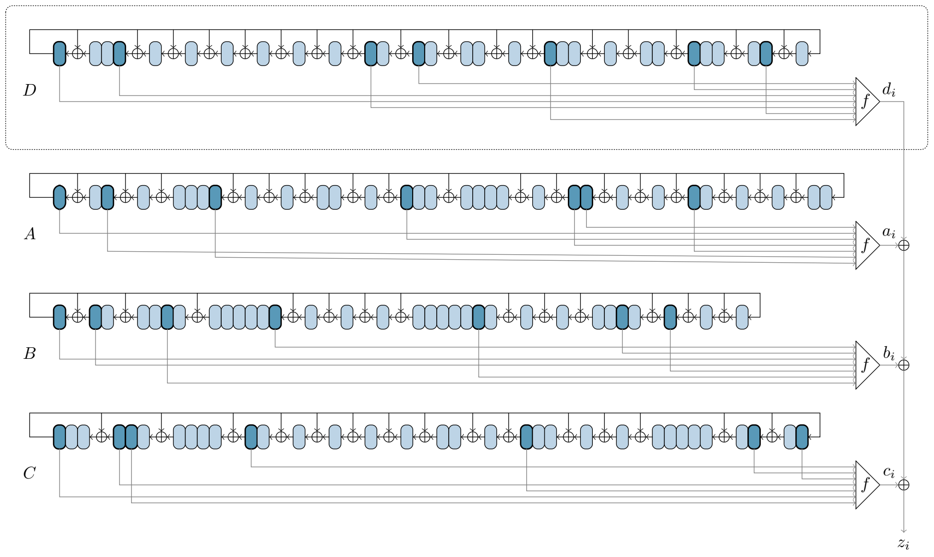

Fig. 3: Overview of the keystream generation of `GEA-1` and `GEA-2`. The D register is only present in `GEA-2`.

Initialization. The cipher is initialized via a non-linear feedback shift register of length 64, denoted as S. This register is filled with 0-bits 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), and a 64-bit secret key K. The initialization starts by clocking S 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 0-bits as input. The feedback function consists of f, xored with the bit that is shifted out and the next bit from the input sequence. See Figure 4 for particular tap positions.

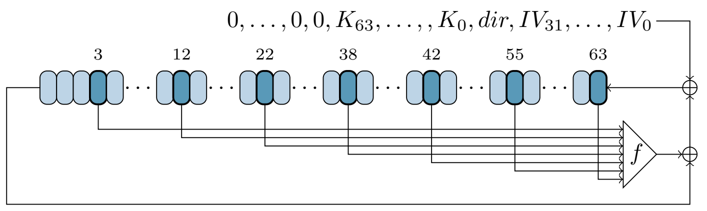
$0, \ldots, 0, 0, K_{63}, \ldots, , K_0, dir, IV_{31}, \ldots, IV_0$
3
12
22
38
42
55
63
f

Fig. 4: Initialization of register S

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 to the all-zero state. 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 $s_0, s_1, \ldots, s_{63}$. 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 5 depicts the process for register B. 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.

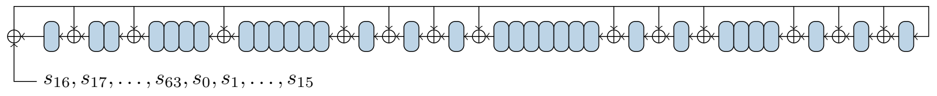

Fig. 5: Initialization of register B

Keystream Generation. When all registers have been initialized, 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 three outputs from the f-functions are xored together to produce one bit of the keystream. Figure 3 shows the particular feedback positions of each register, as well as showing which positions form which input to f. In Figure 3, 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.

2.2 `GEA-2`

The cipher `GEA-2` is a simple extension of `GEA-1`. A fourth register of length 29, called D, is added to the system together with an instance of f. During keystream generation, the output of f from the D register is added to the keystream together with the three others at each clock, as shown in Figure 3. The initialization process of `GEA-2` follows the same mode as for `GEA-1`, but it is done in a longer register that is clocked more times.

Initializing `GEA-2`. As for `GEA-1`, the initialization of `GEA-2` is done via a nonlinear feedback shift register, called W. The length of W is 97, and uses f as its feedback function. The input to `GEA-2` are the same as for `GEA-1`; a 32-bit IV and a direction bit dir that are public, and a secret 64-bit key K.

Initialization starts with W being set to the all-zero state. Next, it is clocked 97 times, inserting one bit from the input sequence for each clock. The order for inserting IV, dir and K is the same as for `GEA-1`. After K_{63} is inserted, W is clocked another 194 times, with 0 as input. This process, together with the particular tap positions for f, is shown in Figure 6.

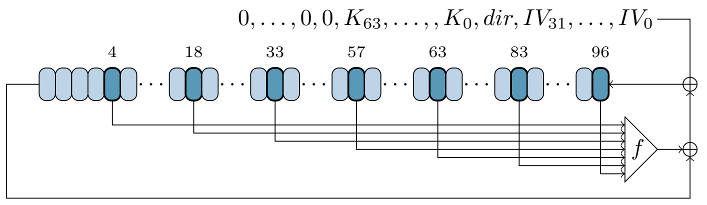
$0, \ldots, 0, 0, K_{63}, \ldots, , K_0, dir, IV_{31}, \ldots, IV_0$
4
18
33
57
63
83
96
f

Fig. 6: Initialization of register W

The content of W is now taken as a 97-bit string $w = w_0, \ldots, w_{96}$, and inserted in A, B, C and D in much the same way as with `GEA-1`. The four registers starts from the all-zero state, and are filled with the bits of w in the same way as shown in Figure 5. The offsets of where in the sequence w each register starts is different than for `GEA-1`. Register D inserts the bits of w in the natural order $w_0, \ldots, w_{96}$, whereas the registers A, B and C start with bits w_{16}, w_{33} and w_{51}, respectively. Again, if any of the registers happens to end up in the all-zero state after initialization, the bit in position 0 is hard-coded to 1 before key generation start.

2.3 Deconstructing the Filter Function

The filter function $f : \mathbb{F}_2^7 \to \mathbb{F}_2$ has a very particular Walsh (or Fourier) spectrum. Namely

$$\widehat{f}(\alpha) = \sum_{x \in \mathbb{F}_2^7} (-1)^{f(x)+\langle \alpha, x \rangle} \in \{0, \pm 2^{\frac{7+1}{2}}\} ,$$

for all $\alpha \in \mathbb{F}_2^7$. Several ways to construct such a Boolean function are known (we refer to Claude Carlet's treatment [9] for a detailed presentation of the required theory of Boolean functions). They appear as component functions of almost bent functions or can be constructed using bent functions in one dimension smaller. While we do not know how f was actually designed, it can certainly be decomposed into two bent functions

$$f_i : \mathbb{F}_2^6 \to \mathbb{F}_2$$

as

$$f(x) = (1 + x_6) f_0(x_0, \ldots, x_5) + x_6 f_1(x_0, \ldots, x_5) .$$

Furthermore, the functions f_i are linearly equivalent to Maiorana-McFarland bent functions [27] (as actually all bent functions in 6 bits are classified in [32]). Indeed, we can decompose f_0 further into

$$f_0(x_0, \ldots, x_5) = g_0(x_0, x_1 + x_2, x_2, x_3, x_4, x_5)$$

where g_0 is a Maiorana-McFarland bent function given as

$$g_0(x_0, \ldots, x_5) = \left\langle \begin{pmatrix} x_2 \\ x_3 \\ x_4 \end{pmatrix}, \begin{pmatrix} x_0 + x_1 x_5 + x_5 \\ x_0 x_1 + x_0 x_5 + x_0 + x_1 + 1 \\ x_0 x_1 + x_5 \end{pmatrix} \right\rangle + h_0(x_0, x_1, x_5) ,$$

where

$$h_0(x_0, x_1, x_5) = x_0x_5 + x_1x_5 + x_1 + x_5 \ .$$

In a similar fashion, f_1 can be written as

$$f_1(x_0, \dots, x_5) = g_1(x_0 + x_2 + x_5, x_1, x_2, x_3, x_4 + x_5, x_5) \ .$$

That is, f_1 is linearly equivalent to g_1 where g_1 is again a Maiorana-McFarland bent function. The function g_1 can be written as

$$g_1(x_0, \dots, x_5) = \left\langle \begin{pmatrix} x_0 \\ x_3 \\ x_4 \end{pmatrix}, \begin{pmatrix} x_1x_5 + x_2x_5 + x_2 \\ x_1x_5 + x_1 + x_2 + 1 \\ x_5 + 1 \end{pmatrix} \right\rangle + h_1(x_0, x_1, x_5) \ ,$$

where

$$h_1(x_0, x_1, x_5) = x_1x_2 + x_1x_5 \ .$$

We like to note that those insights in the filter function do not play any role in our attacks and, for all we know, do not point at any weakness of the cipher. Rather, they indicate that the filter was generated following known and valid principles.

3 An Attack on GEA-1

First we recall some basic facts about LFSRs in Galois mode, as depicted in Figure 7. For further reading we refer to ([34, p. 378 ff.],[20, p. 227]).

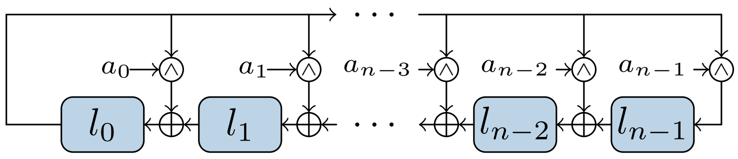

Fig. 7: An LFSR in Galois mode.

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

$$G_L \cdot l \coloneqq \begin{pmatrix} a_0 & 1\ 0 \dots 0 \\ a_1 & 0\ 1 \dots 0 \\ \vdots & \vdots\ \vdots \ \ddots \ \vdots \\ a_{n-2} & 0\ 0 \dots 1 \\ a_{n-1} & 0\ 0 \dots 0 \end{pmatrix} \cdot \begin{pmatrix} l_0 \\ l_1 \\ \vdots \\ l_{n-2} \\ l_{n-1} \end{pmatrix} = \begin{pmatrix} a_0l_0 + l_1 \\ a_1l_0 + l_2 \\ \vdots \\ a_{n-2}l_0 + l_{n-1} \\ a_{n-1}l_0 \end{pmatrix}$$

and the characteristic polynomial of G_L is

$$g(X) \coloneqq X^n + a_0X^{n-1} + \cdots + a_{n-2}X + a_{n-1} \ .$$

Throughout this work, we consider the case in which g is primitive. The characteristic polynomial $g(X)$ is equal to the minimal polynomial of G_L if and only if $a_{n-1} = 1$. Vice versa, given a primitive polynomial $g(X) \coloneqq X^n + a_0 X^{n-1} + \cdots + a_{n-2}X + a_{n-1}$, 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 companion matrix of an LFSR in Galois mode with minimal polynomial g. We call such a matrix the *Galois matrix* and the corresponding minimal polynomial the *Galois polynomial* in the sequel. Moreover, given an LFSR L in Galois mode with minimal (primitive) polynomial g, we denote the Galois matrix with G_g. In the case of `GEA-1` the Galois polynomials are

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

The initialization process of the registers A, B and C with the string s is obviously linear. Hence 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

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

where α, β and γ denote the states of the three LFSRs after the initialization phase. We exclude here the unlikely case that α, β or γ is still in the all-zero state after the shifted insertion of s.

We are now interested in the number of possible starting states of the registers after this initialization. For those considerations, we used the computer algebra system `sagemath` [37]. The corresponding code is attached in Appendix A. The first observation is that all the 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. However, when considering pairs of registers, the picture changes drastically. In particular, the number of possible joint states after initialization of the registers A and C is much smaller than expected. For this it is convenient to consider the kernels of the linear mappings. Clearly, the corresponding linear mappings represented by M_A, M_B and M_C have kernels of dimension of at least 33, 32 and 31, respectively. If we denote $T_{AC} \coloneqq \ker(M_A) \cap \ker(M_C)$ and $U_B \coloneqq \ker(M_B)$ then, curiously enough, we have

1. $\dim(T_{AC}) = 24$ and $\dim(U_B) = 32$,
2. $U_B \cap T_{AC} = \{0\}$.

From this it directly follows that $\mathbb{F}_2^{64}$ can be decomposed into the direct sum $U_B \oplus T_{AC} \oplus V$, 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 $u \in U_B$, $t \in T_{AC}$, $v \in V$ and

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

From this decomposition, s can be computed with a Divide-and-Conquer attack with a complexity[5] of 2^{37} `GEA-1` evaluations to build (and sort) 2^8 tables with 2^{24} entries of size 89 bits and a brute-force step of complexity 2^{40} `GEA-1` evaluations for each new session key $K_0, \ldots, K_{63}$. The details will be given in Section 3.1.

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 the weakness that is highly unlikely to occur unintentionally.

Once s is determined, $K_0, \ldots, K_{63}$ can be recovered as follows. Let S_i denote the state of register S after i clocks of initialization. So $S_0 = (0, 0, \ldots, 0)$ and $S_{225} = (s_0, s_1, \ldots, s_{63})$ where all the s_j are known (see also Figure 4). The last 128 clocks of S all insert 0-bits from the string containing K, dir and IV, so it is straightforward to clock S_{225} backwards 128 times and find the content of S_{97}. Let $S_{97} = (a_0, a_1, \ldots, a_{63})$, where all the a_i are known.

Starting from the other side, the first 33 clocks of S_0 only insert the known bits $IV_0, IV_1, \ldots, IV_{31}, dir$, so the content of S_{33} is fully known. The next clock inserts $K_0 + b_0$ at position 63 of S_{34}, where b_0 is a known bit. Further clocking do not change the content of this cell, but only shifts it further positions to the left, so after 63 clockings starting from S_{34} we have $S_{97} = (K_0 + b_0, \ldots)$. Equating S_{97} from the forward direction with S_{97} from the backward direction gives us $K_0 = a_0 + b_0$.

With K_0 known, position 63 of S_{35} can now be given as $K_1 + b_1$, where b_1 is known. Clocking S_{35} forward 62 times gives $S_{97} = (K_0 + b_0, K_1 + b_1, \ldots)$ and again equating with S_{97} from the backwards side gives $K_1 = a_1 + b_1$. Continuing this way recovers the whole session key K. 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.

[5] 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).

3.1 A Simple Divide-and-Conquer Attack on `GEA-1`

A table *Tab* is built by exhaustive search over the 2^{32} values

$$\beta_{t,v} = M_B(t+v), t \in T_{AC}, v \in V ,$$

plugging $\beta_{t,v}$ into register B and clocking it ℓ times. The parameter ℓ will be determined later. The table is divided into 2^8 sub-tables $Tab[v]$ indexed by v; the output bits $b_{t,v}^{(0)}, \ldots, b_{t,v}^{(\ell-1)}$ (after applying the filter f), together with t, are stored in $Tab[v]$. We then sort each $Tab[v]$ according to $b_{t,v}^{(0)}, \ldots, b_{t,v}^{(\ell-1)}$ interpreted as an ℓ-bit integer. The table has 2^{32} entries of size $\ell + 24$ bits, so it can be generated and sorted with a complexity of $32 \cdot 2^{32} = 2^{37}$ operations if the size of ℓ is negligible (which it is, as we will see below).

Given ℓ bits of keystream z_i, the sequence s is recovered as follows. First, an exhaustive search is conducted over the 2^{40} values

$$\alpha_{u,v} = M_A(u+v),\ \gamma_{u,v} = M_C(u+v),\ u \in U_B,\ v \in V ,$$

plugging $\alpha_{u,v}$ into A, $\gamma_{u,v}$ into C and clocking both registers ℓ times. We denote by $a_{u,v}^{(0)}, \ldots, a_{u,v}^{(\ell-1)}$, resp., $c_{u,v}^{(0)}, \ldots, c_{u,v}^{(\ell-1)}$, the output stream of register A, resp., C after applying the filter f. For each (u, v), the output stream

$$a_{u,v}^{(0)} \oplus c_{u,v}^{(0)} \oplus z_0, \ldots, a_{u,v}^{(\ell-1)} \oplus c_{u,v}^{(\ell-1)} \oplus z_{\ell-1}$$

is generated and it is checked whether there is a match in $Tab[v]$. In the positive case, this gives candidates for u, t and v and finally for $s = u \oplus t \oplus v$ if and only if the entry is found in $Tab[v]$. The overall complexity of this step is 2^{40}, assuming that generating ℓ bits of keystream, together with a search in the sorted table is below the complexity of one `GEA-1` evaluation (for generating a keystream of size 128 bit).

The correct key will always be identified, but this procedure can also suggest some wrong keys, depending on the value of ℓ. There is a trade-off between the amount of known plaintext available, the size of table, and the number of remaining keys. A wrong partial key $u \oplus v$ yields a bitstream stored in $Tab[v]$ with probability $\frac{1}{2^\ell}$ for each entry, if we assume the widely accepted hypothesis that an ℓ bit output of a filtered LFSR behaves like uniformly distributed random bits as long as ℓ is below its period (which will be the case here). We have 2^{24} entries in $Tab[v]$, thus there are at most 2^{24} possible correct entries per partial key. In other words, the probability that a wrong partial key does not cause a hit in $Tab[v]$ is $\left(1 - \frac{1}{2^\ell}\right)^{2^{24}}$ and therefore the probability that none of the wrong partial keys cause a hit is

$$\left(\left(1 - \frac{1}{2^\ell}\right)^{2^{24}}\right)^{(2^{40}-1)} \approx \left(1 - \frac{1}{2^\ell}\right)^{2^{64}} .$$

If we want the probability for no false hit to be larger than or equal to $\frac{1}{2}$, we can choose $\ell = 65$, for which we get $\left(1 - \frac{1}{2^\ell}\right)^{2^{64}} \approx 0.607$. The corresponding size of the table *Tab* with this choice for ℓ is only 44.5 GiB and it can be built in time 2^{37}.

If we only have $n < 65$ known plaintext bits, we obtain a set of roughly 2^{64-n} remaining keys; if $n \geq 24$ we expect less than 2^{40} candidate keys, and can try each of them without increasing significantly the complexity of the attack. The key candidates can be verified using redundancy in the frame (*e.g.* checksums), or known bits in a different frame (*e.g.* headers). We need 65 known bits of information in total, but they can be spread over several frames as long as one frame has 24 bit of known plaintext. In practice, there are many GPRS frames with predictable content, so that intercepting a ciphertext with some known plaintext bits is not an issue (see Section 5.1 for details). Thus the attack is fully practical.

Note that the attack presented is rather straightforward and can probably be optimized. Moreover, several trade-offs are possible. For instance, one can alternatively apply the attack by building a table corresponding to the 2^{40} choices of $\alpha_{u,v}, \gamma_{u,v}$ and then performing an exhaustive search over 2^{32} values for $\beta_{t,v}$. This way, one would need 2^{32} `GEA-1` evaluations as the online time complexity, but much more memory for storing the table. For example, the memory needed to store 2^{40} values of 65-bit length is 8.125 TiB.

On the Likelihood that dim(T_{AC}) = 24. We did an extensive computer search to answer the question if the situation in `GEA-1` is unlikely to occur. To do so, over 10^6 samples, we randomly generated two primitive polynomials g_1, g_2 of degrees d_1, d_2, built the corresponding Galois matrices G_{g_1}, G_{g_2}, computed the representation matrices $M_{G_{g_1}}, M_{G_{g_2},cs}$ for the initialization and computed the dimension of the intersection $T_{G_{g_1},G_{g_2},cs}$. Here, the parameter *cs* denotes the cyclic shift applied in the initialization process of the register. In Tables 1, 2, and 3, the results for the parameters as in `GEA-1` are given, i.e., for the parameters $d_1 = 31$ and $d_2 = 32$, $d_1 = 31$ and $d_2 = 33$, $d_1 = 32$ and $d_2 = 33$ with the corresponding shifts. A Sage program that allows the reader to repeat those experiments is provided in Appendix B.

Table 1: Behavior of intersections for randomly generated LFSRs of lengths $d_1 = 31$, $d_2 = 32$ and $cs = 16$ (10^6 tries)

Dimension of intersection	< 5	5	6	7	8	9	10	11
# of spaces	996,027	3,002	742	171	49	6	1	2

Recall that in `GEA-1`, the intersection is of dimension 24. Thus, in general, our attack is avoided almost automatically when choosing random primitive feedback polynomials and further research needs to be conducted to better understand the design of `GEA-1`.

Table 2: Behavior of intersections for randomly generated LFSRs of lengths $d_1 = 31$, $d_2 = 33$ and $cs = 32$ (10^6 tries)

Dimension of intersection	< 5	5	6	7	8	9	10	11
# of spaces	998,027	1,490	366	86	26	5	0	0

Table 3: Behavior of intersections for randomly generated LFSRs of lengths $d_1 = 32$, $d_2 = 33$ and $cs_1 = 16, cs_2 = 32$ (10^6 tries)

Dimension of intersection	< 5	5	6	7	8	9	10	11
# of spaces	999,065	701	181	39	10	3	1	0

Experimental Verification. In this section we address our C++ implementation of the simple Divide-and-Conquer attack on `GEA-1`.

We first utilized `sage` to generate V and bases for T_{AC} and U_B. We then built a table *Tab* of 2^{32} entries, similarly as described above. Notice that each sub-table $Tab[v]$ is implemented as an array of 2^{19} sorted vectors containing entries consisting of 64 bits for $b_{t,v}$ and 24 bits representing t. The remaining bit of $b_{t,v}$ is implicitly stored as an index in $Tab[v]$. *Tab* is stored on disk such that it can be loaded when the attack gets executed again.

Given 65 bits of keystream z_i, the recovery of the initial state s is implemented as follows. For each combination of $u \in U_B$ and $v \in V$ the output stream is generated using a bitsliced implementation of A and C. To check whether there is a match in *Tab* we search through the vector at $Tab[v][idx]$ where idx represents the 19 most significant bits of the output stream. If there is a match we restore t from *Tab* and return $s = u \oplus t \oplus v$. To speed things up we parallelized both the generation of *Tab* and the recovery of s using OpenMP [10].

To test our implementation we first picked random values for t, u, v. After this we determined suitable values for K, IV and dir by clocking the register S backwards. Then we used the `GEA-1` implementation that we were provided with to generate 65 keystream bits. Finally we checked if our attack restores the correct initial state $s = u \oplus t \oplus v$.

We executed the attack on a cluster made up of four AMD EPYC 7742 64-Core Processors. Generating and storing *Tab* takes 30 minutes whereas loading it from disk only takes five minutes. *Tab* is 46 GiB in size and the recovery of s has a running time of 25 minutes averaged over six runs.

4 An Attack on GEA-2

`GEA-2` does not suffer from the same problems as `GEA-1` for initialization. However, it is still possible to mount an attack on `GEA-2` that does not target initialization, but keystream generation. The idea is to combine a list merging algorithm and algebraic techniques.

4.1 Algebraic Cryptanalysis

The algebraic degree of the filtering function f is 4. The filtering function also has an algebraic immunity of 4. But, as the 4 registers are never mixed, the number of monomials present in the system of equations formed by the relations between the keystream and the initial state is very limited. More precisely, this number is upper bounded by

$$1 + \sum_{i=1}^{4} \binom{29}{i} + \binom{31}{i} + \binom{32}{i} + \binom{33}{i} = 152682\,.$$

This relatively small number would directly imply a powerful attack, just by using a linearisation technique, or, even more powerful, by applying the Berlekamp-Massey algorithm [2,26], as this value is naturally an upper bound to the linear complexity of the output sequence (a direct consequence of Blahut's Theorem [5]).

However, each session in `GEA-2` (or `GEA-1`) is limited to 1600 bytes, that is 12800 bits. This data limitation frustrates direct algebraic cryptanalysis, as the linearization technique is impossible when we have less equations than monomials.

4.2 Guess-and-Determine

The Guess-and-Determine technique seems to have its origin already in the cryptanalysis of A5/1 cipher [18,1]. It can be a powerful technique, specially for analyzing stream ciphers. In the context of algebraic cryptanalysis, it has been shown in [13] that Guess-and-Determine can really help to provide much simpler systems of equations. In a context of general multivariate system solving algorithms, this technique is known as the hybrid approach [3].

For `GEA-2`, we mainly want to reduce the number of monomials present in our system below 12800. By guessing n_d, n_a, n_b and n_c bits in the registers D, A, B and C respectively, we find that the number of non-constant monomials in the equations is upper bounded by

$$\sum_{i=1}^{4} \binom{29-n_d}{i} + \binom{31-n_a}{i} + \binom{32-n_b}{i} + \binom{33-n_c}{i}\,.$$

To get a system of equations of size below 12800 one needs to guess at least 59 bits of the initial state. One choice is $n_d = 29 - 16 = 13$ bits in the first register, $n_a = 31 - 16 = 15$ bits in the second register, $n_b = 32 - 17 = 15$ bits in the third register and $n_c = 33 - 17 = 16$ bits in the fourth register.

This leads to an attack complexity of 2^{59} times the cost of solving a linear system of size 12800, which is much more than the cost of the exhaustive search. We therefore need to combine guessing with other techniques.

4.3 Divide-and-Conquer Technique

The sum of the output of the four filtered registers frustrates the specific Divide-and-Conquer cryptanalysis described by T. Siegenthaler in 1985: correlation attacks [36].

However, Divide-and-Conquer techniques can also be applied when we adapt it for the following problem. We are given two sets S_1 and S_2, as well as two functions $f_1 : S_1 \to \mathbb{F}_2^c$ and $f_2 : S_2 \to \mathbb{F}_2^c$. For a given $t \in \mathbb{F}_2^c$ the problem is to find all $s_1 \in S_1$ and $s_2 \in S_2$ such that $f_1(s_1) + f_2(s_2) = t$. This problem arises quite often in cryptography and started in [35] with the cryptanalysis of the knapsack-based cryptosystem. Since then, advanced solving techniques have shown they can be a powerful tool for the cryptanalyst [6,12,33,24,22].

One way to solve this problem is to use a hash table H. Typically, for all $s_1 \in S_1$ we compute $f_1(s_1)$ and add s_1 to $H[f_1(s_1)]$. Then for each $s_2 \in S_2$ we compute $f_2(s_2)$ and check the corresponding values for s_1 in $H[t + f_2(s_2)]$. Using the right structure for H the complexity of exhausting all the solutions is $\mathcal{O}(|S_1| + |S_2|)$ in time and $\mathcal{O}(|S_1|)$ in memory.

Remark. This algorithm performs $|S_2|$ random accesses to H. If the table is too large to fit in RAM it may be faster to build the two lists, then to sort them and finally to sequentially go through them to find matches.

4.4 Description of the Attack

The techniques involved in our attack do not work in practice when used alone. However, they can be combined in an elegant way to recover the initial state with complexity significantly lower than 2^{64}. Our attack works as follows.

1. Guess $n_a + n_d$ bits in both registers A and D (note that the choice of values for n_a and n_d is not the same as in Section 4.2). The choice of registers A and D has been done with respect to their length, so as the choice of n_a and n_d that lead to the smallest number of guesses.
2. Using linearization technique, derive linear combinations of the keystream bits that are independent of the remaining variables in registers A and D. This corresponds to a set a linear masks m_i, for $1 \le i \le c$, such that for all i, $m_i \cdot s_{A+D}$ is constant, where s_{A+D} denotes the xor-sum of the sequences generated by the registers A and D, and $\cdot$ is the scalar product.
3. Apply the Divide-and-Conquer technique described previously, with S_1 covering all initial states β in register B and S_2 covering all initial states γ in register C, with f_1 and f_2 being defined by the linear masks, and t_i defined as $m_i \cdot z \oplus m_i \cdot s_{A+D}$, where z is the known keystream, and $m_i \cdot s_{A+D}$ is known:

$$f_1 : \beta \mapsto (m_1 \cdot s_B, \ldots, m_c \cdot s_B)$$
$$f_2 : \gamma \mapsto (m_1 \cdot s_C, \ldots, m_c \cdot s_C)$$

First, we build polynomials corresponding to the output of each filtered register, with the initial value of the register bits as variables. We use register C as an example since it is the largest one. Since the LFSR is linear, we can write the state as a matrix representing the linear expression of each bit in terms of the 33 variables; clocking the LFSR is just a matrix product with a total cost of at most $12800 \times 33^3 = 2^{28.8}$. This could probably be improved further, but will be a negligible cost in the attack anyway.

Next, we notice that guessing $n_d = 9$ bits from register D and $n_a = 11$ bits from register A decreases the number of possible non-constant monomials in the 20+20 remaining variables from A and D to

$$\sum_{i=1}^{4} \binom{20}{i} + \binom{20}{i} = 12390$$

which is smaller than the amount of data available per session. Thus we can perform a Gaussian elimination on the system of equations to derive at least $12800 - 12390 = 410$ linear masks m_i on the output of A and D, such that every non-constant monomial vanishes. On a 64-bit computer the cost of this step is around $12800 \times 12390 \times 12390/64 = 2^{34.8}$ simple operations on 64-bit words.

In order to evaluate f_1 and f_2 efficiently, we write them as polynomials in the B and C variables, respectively. To do so we first choose $c = 64$ masks and we compute the corresponding polynomial expressions of outputs from B and C. This corresponds to multiplying a binary matrix of size $12800 \times (\sum_{i=1}^{4} \binom{32}{i} + \binom{33}{i})$ by a binary matrix of size 64×12800. This requires $64 \times 12800 \times 88385/64 = 2^{30.1}$ simple operations on 64-bit words. We also apply the masks to the keystream sequence with a negligible cost.

At the end of the previous step we have 64 equations of the form $P_B^i = P_C^i$ where P_B^i and P_C^i are polynomials in variables from registers B and C respectively and we can apply the Meet-in-the-Middle technique to retrieve the possible values for B and C. First we evaluate $(P_B^0, P_B^1, \ldots, P_B^{63})$ for all the 2^{32} possible initial states of register B and store the result in a hash table H. Then we evaluate $(P_C^0, P_C^1, \ldots, P_C^{63})$ for all the 2^{33} possible starting states of C and get the corresponding values for B by looking into the hash table. Using the enumeration technique of [7], we can evaluate the 64 degree-4 polynomials on all 2^n states for a cost of only $2^n \times 64 \times 4$ bit operation. Therefore, the cost of this step is roughly $(2^{32} + 2^{33}) \times 4 \times 64/64 = 2^{35.6}$ operations on 64-bit words plus $2^{32} + 2^{33} = 2^{33.6}$ random accesses to the hash table.

Finally, for all the remaining values for registers B and C we solve the system of equations in variables from A and D. As it was already echelonized in the first step of the attack we only have to check whether it is consistent or not, requiring approximately 12390 bit-operations. Since there are only a few remaining key candidates, this step is negligible.

Overall the attack requires:

- $2^{20} \times (2^{34.8} + 2^{30.1} + 2^{35.6}) = 2^{56.3}$ operations on 64-bit words
- $2^{20} \times (2^{33.6}) = 2^{53.6}$ memory accesses

In terms of GEA-2 operations, we assume that one encryption requires at least 64 word operations[6], and that one memory access is comparable to an encryption call. Therefore the complexity is equivalent to $2^{53.7}$ GEA-2 encryptions. The memory complexity corresponds to $2^{32} \times 64 = 2^{38}$ bits.

4.5 Improved Attack

We have developed two tricks to decrease the complexity of our attack against GEA-2. The first one is based on the observation that we perform the same computations several times and that this can be avoided by reorganizing them. The second improvement is highly inspired from classical time/data trade-offs where a sequence of n keystream bits can be seen as k (shifted) sequences of $n-k$ keystream bits.

Gaussian Elimination Only Once. The first Gaussian elimination is performed 2^{20} times, once for each guessed value of the 20 chosen bits of registers A and D. But since the polynomials are of degree 4, guessing a variable cannot *create* a monomial of degree 4. Thus, before starting to guess variables, it is possible to partially echelonize the system by removing all degree 4 monomials which do not contain a variable that will be guessed. This removes $\binom{20}{4} + \binom{20}{4} = 9690$ equations and requires $12800 \times (\sum_{i=1}^{4} \binom{29}{i} + \binom{31}{i}) \times 9690/64 = 2^{36.9}$ operations on 64-bit words. Then for each guess the first Gaussian elimination is performed on a matrix with $12800-9690 = 3110$ rows and $\sum_{i=1}^{3} \binom{20}{i} + \binom{20}{i} = 2700$ columns. As a consequence, the time complexity of the attack becomes:

- $2^{36.9} + 2^{20} \times (2^{28.4} + 2^{30.1} + 2^{35.6}) = 2^{55.6}$ operations on 64-bit words
- $2^{20} \times (2^{33.6}) = 2^{53.6}$ memory accesses

Reducing Number of Guesses. We can improve the attack using the classical trick of targeting the internal state at several different clocks, instead of focusing only on the initial state. The novelty here is that we can find masks which simultaneously work for several shifted keystream sequences.

First, we use $n_d = 10$ and $n_a = 11$, so that the number of non-constant monomial from A and D is only $\sum_{i=1}^{4} \binom{19}{i} + \binom{20}{i} = 11230$. We target one of the 753 first internal states, therefore we extract shifted keystream sequences of length 12047 produced by each of those states. The initial state produces keystream $z_0 \cdots z_{12046}$, the state after one clock produces keystream $z_1 \cdots z_{12047}$, and so on. We define V as the vector space of masks m (of length 12047) such that $m \cdot z$ is independent of z for all the 753 sequences considered; V has dimension $12047 - 753 + 1 = 11295$.

Using the strategy of the previous attack, for each guess of the 21 bits in registers A and D, we can deduce a vector space of dimension $12047 - 11230 = 817$ of masks such that $m_i \cdot s_{A+D}$ is constant. We intersect this vector space

[6] In a brute-force search, the initialization requires at least 195 clocking of the W register per key.

with V to obtain a space of dimension 65 of masks such that both $m_i \cdot s_{A+D}$ and $m_i \cdot z$ are constant, and we run the previous attack with 64 independent masks from this space.

The probability that the guess of the 21 bits is correct for at least one of the 753 first internal states is $1 - (1 - 2^{-21})^{753} \approx 2^{-11.4}$. Thus we have to repeat this step with $2^{11.4}$ different guesses on the average, and the time complexity becomes:

- $2^{36.9} + 2^{11.4} \times (2^{27.9} + 2^{30.1} + 2^{35.6}) = 2^{47}$ operations on 64-bit words
- $2^{11.4} \times (2^{33.6}) = 2^{45}$ memory accesses

This is equivalent to roughly $2^{45.1}$ `GEA-2` encryptions.

4.6 Recovering the Master Key

This attack recovers the internal states of the registers A, B, C and D, either at the beginning or after a few clocks. From this we can easily recover the sequence w, because the initialization and the update of the LFSRs are linear functions. As in the case of `GEA-1`, we can also recover the master key by clocking the W register forwards from the zero state, and backwards from the recovered state w. Therefore, we only have to perform the attack once per GPRS session; we can decrypt all the messages in a session if we have one known message of length 1600 byte.

4.7 Using Less Data

Our attack can be applied with less data than 12800 bits of keystream. In that case the time complexity is increased as shown in Figure 8. To reach a complexity below 2^{64} (the complexity of an exhaustive search on the key), we need around 1468 consecutive keystream bits.

4.8 Experimental Verification

We now briefly describe our proof of concept implementation of the attack on `GEA-2`. The implementation consists of a `sage` and C++ part which is made accessible to `sage` using Cython.

In an initial step we built matrices that represent polynomials corresponding to the filtered output of B and C by evaluating B and C symbolically in `sage`. Here we enumerated the 12 most significant bits and therefore we do not have one but 4096 matrices for each register. This allows straightforward parallelization of the Divide-and-Conquer step when using the enumeration technique of [7]. The matrices are stored on disks such that they can be loaded when the attack gets executed again.

Given the keystream we first compute V as described above. Then we start guessing 21 bits in A and D. For each guess we compute 64 corresponding masks. These are then handed over to the C++ part which builds upon the M4RI [25]

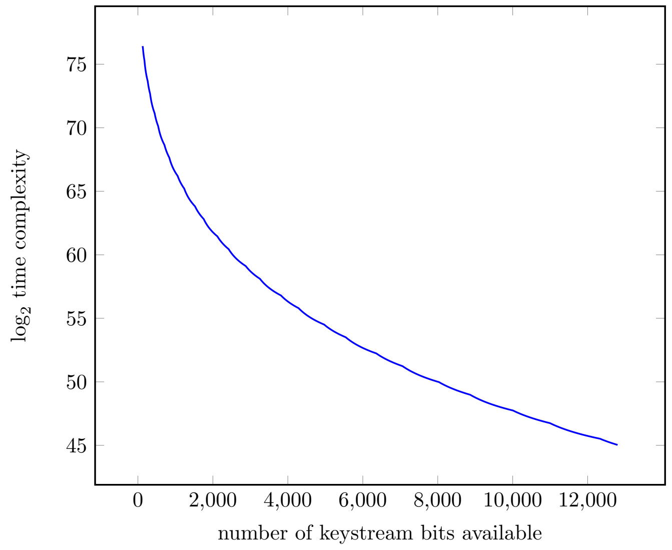

Fig. 8: Time complexity of our attack against `GEA-2` as a function of the number of consecutive keystream bits available.

library to apply the Divide-and-Conquer technique. Candidates for β and γ are returned to `sage` to check if they lead to consistent solutions for the remaining bits in registers A and D.

To test our implementation we picked random values for K, IV and dir and computed the keystream with the `GEA-2` implementation we were provided with. We also determined the first 753 internal states such that we can directly check whether a guess of the 21 bits in A and D was correct or not.

We executed the attack on the same hardware as for the `GEA-1` attack. It takes about one hour to perform the calculations on one guess and therefore we get roughly four months as the runtime of a full attack.

5 Discussion

In the following, we discuss the real-world attack feasibility, the attack severity and the attack implications.

5.1 Attack Requirements

To recover the full session key of a `GEA-1` encrypted connection, the attacker must meet the following conditions: The attacker must *i*) sniff the encrypted radio traffic of the victim's phone and *ii*) know 65 bits of the keystream, preferably at the beginning of the keystream. As shown by Nohl and Melette [29], sniffing the encrypted traffic can be conducted with the osmocom-bb project [31] using ordinary hardware.

Meeting the requirement of knowing 65 bits of the keystream can be achieved by exploiting predictable SNDCP (Subnetwork Dependent Convergence Protocol) and IP header patterns. A GPRS data connection encapsulates each IP packet with the SNDCP, which is then encrypted by the LLC (Logical Link Control) protocol. In a small experiment, we study patterns that remain stable and predictable over multiple GPRS connections in the SNDCP and IP header. Header fields like the SNDCP NSAPI, the IP Version, TTL, ToS, and destination IP address fields remain stable over multiple connections. Consequently, guessing 65 plaintext bits and obtaining 65 keystream bits is plausible by an entirely passive attacker.

In contrast, the attack on `GEA-2` requires the attacker to know the whole 1600 bytes of keystream to recover the session key with complexity $2^{45.1}$ `GEA-2` evaluations. Accordingly, the attacker must correctly predict 1600 bytes of plaintext. Depending on the attacker's capabilities, this can be within the area of possibility. If the attacker controls a server that the victim visits, he can access the bytes sent or receives, and consequently, the attack can predict 1600 bytes. The recovered key is then valid for the whole GRPS session, including other traffic of interest. Such an attack may require some social engineering, e. g., a phishing attack, to convince the victim to visit the website.

5.2 Attack Severity

In GPRS the operator chooses the encryption algorithm, i. e., `GEA-1`, `GEA-2`, `GEA-3` (based on KASUMI with a 64-bit key), or `GEA-4` (based on KASUMI with a 128-bit key). According to a study by Tomcsányi et al. [11], that analyzes the use of the ciphering algorithm in GRPS of 100 operators worldwide, most operators prioritize the use of `GEA-3` (58) followed by the non-encrypted mode `GEA-0` (38). Only a few operators rely on `GEA-2` (4), while no operator uses `GEA-1` (0). Consequently, the likelihood for an attack based on the `GEA-1` and `GEA-2` vulnerabilities is *nowadays* comparably small.

To draw a complete picture, we additionally analyze the support of both algorithms in mobile phones. Since 2013, ETSI prohibits implementing `GEA-1` in mobile stations, while `GEA-2` and the non-encrypted mode (`GEA-0`) are still mandatory to be implemented [16]. We tested a range of current phones if they follow the specification regarding disabling the `GEA-1` cipher. We use an osmocom GPRS setup which we extended with the support of `GEA-1` and `GEA-2` [31]. Table 4 shows a selection of phones in which we cover a wide range of baseband manufacturers. Those manufacturers are responsible for implementing the standard accordingly. Surprisingly, all tested phones support the vulnerable ciphers `GEA-1`, thereby clearly disrespecting the specification.

Once the key is recovered, the attacker can decrypt all traffic for the complete GPRS session until the key gets invalid, which happens in the GPRS authentication and ciphering procedure triggered by the network. The start of this procedure depends on the operator's policy. Usually, the procedure starts on an expired timer, e. g., 1-24h, or the change of a location area, which is a regional group of base stations.

Table 4: Overview of the phones and basebands supporting (●) GEA-X

Phone	**Year**	**Baseband**	**GEA-1**	**GEA-2**
Apple iPhone XR	2018	Intel XMM 7560	●	●
Apple iPhone 8	2017	Intel XMM 7480	●	●
Samsung Galaxy S9	2018	Samsung Exynos 9810	●	●
HMD Global Nokia 3.1	2018	Mediatek MT6750	●	●
Huawei P9 lite	2016	HiSilicon Kirin 650	●	●
OnePlus 6T	2018	Qualcomm Snapdragon 845	●	●

5.3 Attack Implications

`GEA-1` provides 40-bit security and is breakable by today's standard hardware. This fact causes severe implications for our mobile Internet connection during the early 2000s and now.

During the early 2000s, Internet connections were barely secured by any transport layer security, such as TLS. Under the assumption that an operator used `GEA-1` for the network, the entire traffic was accessible to a passive attacker. In contrast, nowadays connections are mostly secured by TLS. However, if the network encryption can be bypassed (as with `GEA-1`), metadata is still accessible, such as DNS requests, IP addresses, and hostnames when using the TLS SNI extension. Consequently, the use of `GEA-1` has still far-reaching consequences on the user's privacy and should be avoided at all costs.

Even if the operator uses a stronger cipher like `GEA-3`, the support of `GEA-1` by the phone allows an attacker to recover a previous session key. A requirement for this attack is that the operator also relies on GSM authentication. GSM authentication is not replay protected, and thus the attacker can replay the previous authentication request with a fake base station and instruct the phone to use the vulnerable cipher (Authentication and Ciphering Request). After a complete attack procedure, sending the ciphering request forces the phone to use a weak cipher (i.e. `GEA-1`) for the next data uplink packet. At that point, the attacker can guess the plaintext to recover parts of the keystream and thus also the previous session key. Consequently, the attacker can decrypt the previous session, which was encrypted with a stronger cipher, e. g., `GEA-3`. This shows that even when operators do not actively use `GEA-1`, the weak `GEA-1` design affects the security of today's communication.

Time/Memory Trade-Off Attack against `GEA-2`. While the present attack against `GEA-2` has a complexity of 2^{45} `GEA-2` evaluations and requires a large amount of known keystream, we could also think of a time/memory trade-off attack against `GEA-2`. However, in contrast to A5/1 where this could be applied [21], the initial state of 125 bits prohibits any such attack aiming for the initial state. Building a time/memory trade-off, using e.g. rainbow-tables, (see [30]) targeting directly the 64 bits secret key would only work for a fixed IV. While it would indeed reduce the amount of known keystream needed, it turns the attack into a chosen IV attack, which limits its practical interest.

5.4 Responsible Disclosure and Industry Implications

Following the guidelines of responsible disclosure, we have disclosed the vulnerability to the GSMA and ETSI Coordinated Vulnerability Disclosure programme [19] [14]. We, thereby, followed two aims: In short term, we want to disable the support of `GEA-1` from all mobile phones and thereby restore the specifications conformity. For mitigating the mid-risk of exploiting the `GEA-2` vulnerabilities, we advocate for removing the support of `GEA-2` from the specification.

The main objective of the GSMA CVD program was to disable the support of `GEA-1`. The GSMA informed the affected baseband vendors, phone manufacturers, including Google and Apple, through the CVD program. Further, the GSMA liaised with GCF (Global Certification Forum) [17], the mobile industry's globally recognized certification scheme for mobile phones and wireless devices based on 3GPP standards. The GCF included two test cases as part of version 3.81.0 of the certification criteria, which became available for certification in January 2021. These are: Conformance test case 44.2.5.2.5 Ciphering mode / Non-support of `GEA-1` from 3GPP TS 51.010-1 and field trial test case 5.6.5 GPRS functionality – Non-support of `GEA-1` from GSMA TS.11. Those test cases now allow to verify that the support of `GEA-1` is *disabled* by devices before entering the market.

In contrast, the submission to the ETSI CVD program followed the mid-term goal to remove the support of `GEA-2` from the specification and consequently also from mobile devices. At the time of paper finalization, the ETSI has accepted our CVD submission and considers whether any standards related measures need to be taken. Specification changes usually require the consent of several parties and take accordingly longer. We will continue to fight for removing the support of `GEA-2` from the specification.

6 Conclusion

We have shown that the first version of the GPRS Encryption Algorithm, `GEA-1`, only offers 40-bit (out of 64) security. We have further shown that it is very unlikely for a random instance to suffer from such weaknesses. Since `GEA-1` was designed to be exportable within the export restrictions in European countries in the late 1990s, this might be an indication that a security level of 40 bits was a barrier for cryptographic algorithms to obtain the necessary authorizations. Ultimately, the weak design of `GEA-1` brings security problems for today's communication, even if it is not being actively used by the operators.

The successor algorithm `GEA-2` seems to be a stronger design, in which the weaknesses of `GEA-1` are not present anymore. Still, the cipher does not offer full 64-bit security and we have shown an attack of complexity $2^{45.1}$ `GEA-2` evaluations. Although such an attack is more difficult to be applied in practice, we think that `GEA-2` does not offer a high enough security level for todays standards. Therefore, we strongly recommend that only the much more secure GPRS Encryption Algorithms, starting from `GEA-3`, should be implemented.

Acknowledgment

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, and by French Agence Nationale de la Recherche (ANR), under grant ANR-20-CE48-0017 (project SELECT). Patrick Derbez was supported by the French Agence Nationale de la Recherche through the CryptAudit project under Contract ANR-17-CE39-0003.

Most of all, we give thanks to Dieter Spaar and Harald Welte for their support and contact persons of the osmocom project.

References

1. Anderson, R.J.: A5 (was hacking digital phones). Newsgroup Communication (1994), `http://yarchive.net/phone/gsmcipher.html`, (accessed March 4, 2021)
2. Berlekamp, E.R.: Algebraic coding theory. McGraw-Hill series in systems science, McGraw-Hill (1968), `http://www.worldcat.org/oclc/00256659`
3. Bettale, L., Faugère, J., Perret, L.: Hybrid approach for solving multivariate systems over finite fields. J. Mathematical Cryptology **3**(3), 177–197 (2009), `https://doi.org/10.1515/JMC.2009.009`
4. Biryukov, A., Gong, G., Stinson, D.R. (eds.): Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers, Lecture Notes in Computer Science, vol. 6544. Springer (2011), `https://doi.org/10.1007/978-3-642-19574-7`
5. Blahut, R.E.: Theory and practice of error control codes. Addison-Wesley (1983)
6. Bogdanov, A., Rechberger, C.: A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In: Biryukov et al. [4], pp. 229–240, `https://doi.org/10.1007/978-3-642-19574-7_16`
7. Bouillaguet, C., Chen, H., Cheng, C., Chou, T., Niederhagen, R., Shamir, A., Yang, B.: Fast exhaustive search for polynomial systems in F_2. In: CHES. Lecture Notes in Computer Science, vol. 6225, pp. 203–218. Springer (2010)
8. Brookson, C.: GPRS Security. `https://web.archive.org/web/20120914110208/http://www.brookson.com/gsm/gprs.pdf` (snapshot of September 14, 2012) (2001)
9. Carlet, C., Crama, Y., Hammer, P.L.: Boolean functions for cryptography and error-correcting codes. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering, pp. 257–397. Cambridge University Press (2010), `https://doi.org/10.1017/cbo9780511780448.011`
10. Dagum, L., Menon, R.: Openmp: an industry standard api for shared-memory programming. IEEE computational science and engineering **5**(1), 46–55 (1998)

11. Domonkos P. Tomcsányi, Marco Weyres, Pedro Simao: Analysis of EGPRS Ciphering Algorithms used Worldwide. https://www.umlaut.com/en/analysis-of-egprs-ciphering-algorithms-used-worldwide, (to appear)
12. Dunkelman, O., Sekar, G., Preneel, B.: Improved meet-in-the-middle attacks on reduced-round DES. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) Progress in Cryptology - INDOCRYPT 2007, 8th International Conference on Cryptology in India, Chennai, India, December 9-13, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4859, pp. 86–100. Springer (2007), https://doi.org/10.1007/978-3-540-77026-8_8
13. Duval, S., Lallemand, V., Rotella, Y.: Cryptanalysis of the FLIP family of stream ciphers. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9814, pp. 457–475. Springer (2016), https://doi.org/10.1007/978-3-662-53018-4_17
14. ETSI: ETSI — Coordinated Vulnerability Disclosure). https://www.etsi.org/standards/coordinated-vulnerability-disclosure, (accessed March 4, 2021)
15. 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 October 8, 2020) (1998)
16. ETSI: Digital cellular telecommunications system (phase 2+) (gsm); security related network functions (3gpp ts 43.020 version 15.0.0 release 15). Technical Specification. Available at https://www.etsi.org/deliver/etsi_ts/143000_143099/143020/15.00.00_60/ts_143020v150000p.pdf (accessed October 8, 2020) (2018)
17. GCF: GCF — Global Certification Forum. https://www.globalcertificationforum.org/, (accessed March 4, 2021)
18. Golic, J.D.: Cryptanalysis of alleged A5 stream cipher. In: Fumy, W. (ed.) Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding. Lecture Notes in Computer Science, vol. 1233, pp. 239–255. Springer (1997), https://doi.org/10.1007/3-540-69053-0_17
19. GSMA: GSMA — Coordinated Vulnerability Disclosure Programme. https://www.gsma.com/security/gsma-coordinated-vulnerability-disclosure-programme/, (accessed March 4, 2021)
20. Hoffman, K., Kunze, R.A.: Linear Algebra. PHI Learning (2004), http://www.worldcat.org/isbn/8120302702
21. Kalenderi, M., Pnevmatikatos, D.N., Papaefstathiou, I., Manifavas, C.: Breaking the GSM A5/1 cryptography algorithm with rainbow tables and high-end FPGAS. In: Koch, D., Singh, S., Tørresen, J. (eds.) 22nd International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, August 29-31, 2012. pp. 747–753. IEEE (2012), https://doi.org/10.1109/FPL.2012.6339146
22. Khovratovich, D., Naya-Plasencia, M., Röck, A., Schläffer, M.: Cryptanalysis of *Luffa* v2 components. In: Biryukov et al. [4], pp. 388–409, https://doi.org/10.1007/978-3-642-19574-7_26
23. Koops, B.J.: Crypto law survey. http://www.cryptolaw.org (accessed October 8, 2020) (2013)
24. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound distinguishers: Results on the full whirlpool compression function. In: Matsui, M. (ed.)

Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5912, pp. 126–143. Springer (2009), https://doi.org/10.1007/978-3-642-10366-7_8

25. Martin Albrecht and Gregory Bard: The M4RI Library. The M4RI Team (2021), http://m4ri.sagemath.org (accessed March 4, 2021)
26. Massey, J.L.: Shift-register synthesis and BCH decoding. IEEE Trans. Inf. Theory **15**(1), 122–127 (1969), https://doi.org/10.1109/TIT.1969.1054260
27. McFarland, R.L.: A family of difference sets in non-cyclic groups. J. Comb. Theory, Ser. A **15**(1), 1–10 (1973), https://doi.org/10.1016/0097-3165(73)90031-9
28. MediaTek: Test Vector GEA1/2 — MediaTek-HelioX10-Baseband. https://github.com/Dude100/MediaTek-HelioX10-Baseband/blob/591772a0d659ef0f7bba1953d18f8fe7c18b11de/(FDD)MT6795.MOLY.LR9.W1423.MD.LWTG.MP.V24/driver/cipher/include/gcu_ut.h, (accessed March 4, 2021)
29. Nohl, K., Melette, L.: GPRS intercept: Wardriving your country. Chaos Communication Camp, 2011. Slides available at http://events.ccc.de/camp/2011/Fahrplan/attachments/1868_110810.SRLabs-Camp-GRPS_Intercept.pdf (accessed October 8, 2020). Recorded talk available at https://media.ccc.de/v/cccamp11-4504-gprs_intercept-en#t=1744 (accessed October 8, 2020) (2011)
30. Oechslin, P.: Making a faster cryptanalytic time-memory trade-off. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 617–630. Springer (2003), https://doi.org/10.1007/978-3-540-45146-4_36
31. osmocom: osmocom — Cellular Network Infrastructure. https://osmocom.org/projects/cellular-infrastructure, (accessed March 4, 2021)
32. Rothaus, O.S.: On "bent" functions. J. Comb. Theory, Ser. A **20**(3), 300–305 (1976), https://doi.org/10.1016/0097-3165(76)90024-8
33. Sasaki, Y.: Meet-in-the-middle preimage attacks on AES hashing modes and an application to Whirlpool. In: Joux, A. (ed.) Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6733, pp. 378–396. Springer (2011), https://doi.org/10.1007/978-3-642-21702-9_22
34. Schneier, B.: Applied cryptography - protocols, algorithms, and source code in C, 2nd Edition. Wiley (1996), http://www.worldcat.org/oclc/32311687
35. Schroeppel, R., Shamir, A.: A T=O($2^{n/2}$), S=O($2^{n/4}$) algorithm for certain np-complete problems. SIAM J. Comput. **10**(3), 456–464 (1981), https://doi.org/10.1137/0210033
36. Siegenthaler, T.: Decrypting a class of stream ciphers using ciphertext only. IEEE Trans. Computers **34**(1), 81–85 (1985), https://doi.org/10.1109/TC.1985.1676518
37. The Sage Developers: SageMath, the Sage Mathematics Software System (2020), https://www.sagemath.org

Appendix A Source Code to Compute the Kernels

Listing 1.1: gea1_kernels.sage

```
def getInitMatrix(p,keyLength,shift):
    P.<x> = PolynomialRing(GF(2))
    l = p.degree()
    #construct transformation matrix A for LFSR in Galois mode
    A = companion_matrix(p, "left")
    M = zero_matrix(GF(2),keyLength,l)
    for c in range(keyLength):
        x = zero_vector(GF(2),l)
        k = zero_vector(GF(2),keyLength)
        k[c] = 1
        for j in range(keyLength):
            x[0] = x[0] + k[(j+shift) % keyLength]
            x = A*x
        M[c] = x
    return M

#for GEA-1
P.<x> = PolynomialRing(GF(2))
keyLength = 64
pA = 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
shiftA = 0
pB = x^32+x^31+x^29+x^25+x^19+x^18+x^17+x^16+x^9+x^8+x^7+x^3
+x^2+x+1
shiftB = 16
pC = 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
shiftC = 32

MA = getInitMatrix(pA,keyLength,shiftA)
MB = getInitMatrix(pB,keyLength,shiftB)
MC = getInitMatrix(pC,keyLength,shiftC)

U_B = MB.kernel()
T_AC= MA.kernel().intersection(MC.kernel())
print(U_B.dimension())          #has dimension 32
print(T_AC.dimension())         #has dimension 24
print(T_AC.intersection(U_B).dimension()) #has dimension 0
```

Appendix B Source Code to Compute the Dimensions

Listing 1.2: random_kernels.sage

```
set_random_seed()
P.<x> = PolynomialRing(GF(2))

def get_random_primitive(l):
  V = VectorSpace(GF(2),l)
  v = list(V.random_element())
  p = P(v+[1])
  while (not p.is_primitive()):
    v = list(V.random_element())
    p = P(v+[1])
  return p

#parameters to set
keyLength = 64
l1 = 31
l2 = 33
shift1 = 0
shift2 = 32
samples = 1000000

dim = [0]*40
for i in range(samples):
  #get random primitive polynomials p1 and p2
  p1 = get_random_primitive(l1)
  p2 = get_random_primitive(l2)
  M1 = getInitMatrix(p1,keyLength,shift1)
  M2 = getInitMatrix(p2,keyLength,shift2)
  T = M1.kernel().intersection(M2.kernel())
  dim[T.dimension()] = dim[T.dimension()]+1
  if (((i+1)%1000)==0):
    print('runs␣=',i+1)
    print(dim)
```