LIZARD - A Lightweight Stream Cipher for Power-constrained Devices

Matthias Hamann[1], Matthias Krause[1], and Willi Meier[2]

[1] University of Mannheim, 68131 Mannheim, Germany
{hamann,krause}@uni-mannheim.de
[2] FH Nordwestschweiz, 5210 Windisch, Switzerland
willi.meier@fhnw.ch

Abstract. Time-memory-data (TMD) tradeoff attacks limit the security level of many classical stream ciphers (like E_0, A5/1, Trivium, Grain) to $\frac{1}{2}n$, where n denotes the inner state length of the underlying keystream generator. In this paper, we present LIZARD, a lightweight stream cipher for power-constrained devices like passive RFID tags. Its hardware efficiency results from combining a Grain-like design with the $FP(1)$-mode, a recently suggested construction principle for the state initialization of stream ciphers, which offers provable $\frac{2}{3}n$-security against TMD tradeoff attacks aiming at key recovery. LIZARD uses 120-bit keys, 64-bit IVs and has an inner state length of 121 bit. It is supposed to provide 80-bit security against key recovery attacks. LIZARD allows to generate up to 2^{18} keystream bits per key/IV pair, which would be sufficient for many existing communication scenarios like Bluetooth, WLAN or HTTPS.

Keywords: Stream Ciphers, Lightweight Cryptography, Time-Memory-Data Tradeoff Attacks, $FP(1)$-mode, Grain, RFID

1 Introduction

Stream ciphers have a long history when it comes to protecting digital communication. In 1987, Ronald L. Rivest designed RC4 [49], which was later used in SSL/TLS [21] and the wireless network security protocols WEP [1] and TKIP (often called WPA) [2]. Other well-known stream cipher examples are E_0 of the Bluetooth standard [51] and A5/1 of GSM [15]. Unfortunately, E_0 and A5/1 have been shown to be highly insecure (see, e.g., [42] and [9]) and RC4 also shows severe vulnerabilities, which led to its removal from the TLS protocol [46] and rendered other protocols like WEP insecure [26]. In 2004, the eSTREAM project [45] was started in order to identify new stream ciphers for different application profiles. In the hardware category, aiming at devices with restricted resources, three ciphers are still part of the eSTREAM portfolio after the latest revision in 2012: Grain v1 [34], MICKEY 2.0 [8] and Trivium [17].

Grain v1 uses 80-bit keys, 64-bit IVs and the authors do not give an explicit limit on the number of keystream bits that should be generated for each key/IV

pair.[1] MICKEY 2.0 uses 80-bit keys, IVs of variable length up to 80 bit and the maximum amount of keystream bits for each key/IV pair is 2^{40}. Trivium uses 80-bit keys, 80-bit IVs and at most 2^{64} keystream bits should be generated for each key/IV pair.

Interestingly, all three ciphers of the eSTREAM hardware portfolio are obviously designed for potentially very large keystream sequences per key/IV pair. In contrast, the aforementioned transmission standards all use much smaller packet sizes. For example, A5/1 produces only 228 keystream bits per key/IV pair, where the session key is 64 bits long and the IV corresponds to 22 bits of the publicly known frame (i.e., packet) number.[2] Similarly, Bluetooth packets contain at most 2790 bits for the so-called basic rate. The Bluetooth cipher E_0 takes a 128-bit session key and uses 26 bits of the master's clock, which is assumed to be publicly known, as the packet-specific IV. For wireless local area networks (WLANs), the currently active IEEE 802.11-2012 standard [3] implies that at most 7943 bytes are encrypted under the same key/IV pair using CCMP.[3] Another widespread example for encryption on a per-packet basis is SSL/TLS, which underlies HTTPS and thus plays a vital role in securing the World Wide Web. In the most recent version, TLS 1.2 [21], the maximum amount of data encrypted under the same key/IV pair is $2^{14} + 2^{10}$ bytes (as long as RC4 is not used[4]).

Considering the above examples, the natural question arises whether a stream cipher with attractive features regarding security and efficiency could be designed by specifically targeting such *packet mode* scenarios, where only moderately long pieces of keystream are generated under the same key/IV pair. In this paper, we answer the question in the affirmative by presenting Lizard, a lightweight stream cipher for power-constrained devices. Lizard takes 120-bit keys, 64-bit IVs and generates up to 2^{18} keystream bits per key/IV pair.[5] It is designed to provide 80-bit security against key recovery attacks including generic time-memory-data (TMD) tradoff attacks. This is remarkable insofar as Lizard's inner state is only 121 bit wide. In contrast, Trivium and Grain v1, for example, have an inner state length of at least twice the size of the targeted security level against key recovery attacks. This is due to the inherent vulnerability of classical stream ciphers (i.e., stream ciphers which compute the keystream based on a so-called *initial state*) against TMD tradeoff attacks like those of Babbage

[1] However, in [34], the designers of Grain point out that the feedback polynomial of the employed 80-bit LFSR is primitive and in [33] they state that "[t]he LFSR guarantees a minimum period for the keystream", suggesting a corresponding upper bound on the number of keystream bits allowed per key/IV pair.

[2] More precisely, 114 bits are used for uplink and 114 bits are used for downlink.

[3] In IEEE 802.11-2012, WEP and TKIP are marked as deprecated. The maximum of 7943 bytes per key/IV pair is only achieved when MSDU aggregation is used. Without this type of frame aggregation, the maximum amount of data encrypted with CCMP under the same key/IV pair is 2330 bytes.

[4] For security reasons, RC4 is now forbidden for all TLS versions by RFC 7465 [46].

[5] Note that a maximum packet size of 2^{18} bits covers all of the real-world protocol examples given in the previous paragraph.

[6] and Biryukov and Shamir [13], which allow to compute the secret initial state on the basis of $\mathcal{O}(2^{n/2})$ keystream bits in time and space $\mathcal{O}(2^{n/2})$, where n denotes the inner state length of the underlying keystream generator (KSG). As the state initialization algorithm, which computes the initial state from a given key/IV pair, is efficiently invertible for Trivium and Grain v1, knowing the initial state immediately reveals the secret key.[6] LIZARD, on the other hand, represents the first practical instantiation of the $FP(1)$-*mode* introduced in [32], which provides a method for designing stream ciphers in such a way that, for packet mode scenarios, a beyond-the-birthday-bound security level of $\frac{2}{3}n$ w.r.t. generic TMD tradeoff attacks aiming at key recovery can be proved.[7]

The design of LIZARD is closely related to that of Grain v1, which turned out to be the most hardware efficient member of the eSTREAM portfolio and, hence, can be considered as a benchmark for new hardware-oriented designs. When compared to Grain v1, the major differences of LIZARD are the smaller inner state (121 bit vs. 160 bit), the larger key (120 bit vs. 80 bit) and the fact that LIZARD introduces the secret key not only once but twice during its state initialization. All of these modifications are a direct consequence of implementing the $FP(1)$-mode as explained in section 3.5. Naturally, reducing the size of the inner state also required to change the underlying feedback shift registers and, for security reasons, to choose a heavier output function.

Note that, very recently, another line of research which also pursues the goal of reducing the inner state size of stream ciphers has emerged. In 2015, Armknecht and Mikhalev suggested Sprout [5], which has a Grain-like structure and uses two 40-bit feedback shift registers. Compared to conventional stream ciphers like Grain v1 (and also compared to LIZARD), the characteristic difference of Sprout is that the 80-bit key is not only accessed during the state initialization but also continuously used as part of the state update during the subsequent keystream generation phase.[8] Even though Sprout was broken shortly after publication (see, e.g., [40], [53], [24]), it has sparked interest in the underlying design principle and related ciphers like Fruit [28] have been suggested since. Unfortunately, though being elegant in theory, continuously accessing the secret key often comes at a heavy price in practice. For example, if the key is stored in an EEPROM, the corresponding access times may significantly slow down the operation speed of the KSG. This is particularly true if the key bits are not accessed sequentially (as in the case of Sprout) but at random positions (as in the case of Fruit).[9] Another drawback to continuously reading key bits

[6] Even if the state initialization algorithm is not efficiently invertible, variants of such TMD tradeoff attacks (e.g., aiming at some other inner state occurring during the state initialization) can often be used to recover the secret key. See [32] for an example of such an attack against E_0.

[7] TMD tradoff attacks aiming at recovering some initial state still work for LIZARD, but they do not allow to straightforwardly derive the underlying secret key (see Sec. 4.2 for details).

[8] A similar idea was already used in [19] for the fixed-key variant KTANTAN of the block cipher KATAN.

[9] In fact, Fruit needs to access six different, non-sequential key bits per clock cycle.

from an EEPROM is the associated increase in power consumption. This is especially true for low-cost RFID tags, where the power budget is often as low as 10 μW (cf. [36]).[10] A way to circumvent the problem of continuously accessing the key in the EEPROM might be to buffer it in volatile memory. This, however, would significantly increase the hardware costs of the corresponding implementation due the additionally required flip-flops. On the other hand, in a scenario where the key is fixed (e.g., via burning fuses), continuously accessing key bits (in potentially random order) is clearly feasible. There, however, it might be problematic that Fruit requires each key to be used with less than 2^{15} different IVs, which places a strong limit on the lifetime of corresponding devices in a fixed-key scenario. Finally, note that, unlike Lizard, neither Sprout nor Fruit offer *provable* security against TMD tradoff attacks aiming at key recovery.

We would like to point out that the above arguments by no means imply that we reject the idea of stream ciphers that continuously access the secret key. However, they illustrate the need for alternative ideas that allow to reduce the size of the inner state even in scenarios where continuous key access is not feasible.

Lizard was designed with low-cost scenarios like passively powered RFID tags in mind. In particular, it outperforms Grain v1 in important hardware metrics like cell area and power consumption. Most notably, the estimated power consumption is about 16 percent below that of Grain v1, making Lizard particularly suitable for power-constrained devices. This shows that in scenarios where plaintext packets of moderate length are to be encrypted under individual IVs, the $FP(1)$-mode provides an interesting alternative to conventional state initialization algorithms of stream ciphers.

Structure of the paper: In section 2, we specify the components of Lizard (Sec. 2.1) and describe how it is operated during state initialization (Sec. 2.2) and keystream generation (Sec. 2.3). Building on this, section 3 then provides the corresponding design considerations. In particular, subsection 3.5 contains a detailed explanation of the generic $FP(1)$-mode and how it was implemented for Lizard. Section 4 treats the cryptanalysis of Lizard and in section 5, we present the details of our hardware implementation along with a comparison of the corresponding performance metrics between Lizard and Grain v1. Section 6 concludes the paper and provides an outlook on potential future work. Test vectors can be found in appendix A.

2 Design Specification

The design of Lizard is similar to (and was in fact inspired by) that of the Grain family [33] of stream ciphers. In particular, the inner state of Lizard is

[10] For comparison, Ranasinghe and Cole state in [18] that "[a] tag performing a read operation will require about 5 μW – 10 μW, while a tag attempting to perform a write operation to its E^{2}PROM will require about 50 μW or more." Our entire hardware implementation of Lizard, on the other hand, has an estimated power consumption of only 2.1 μW (cf. Sec. 5).

distributed over two interconnected feedback shift registers (FSRs) as depicted in Fig. 1.

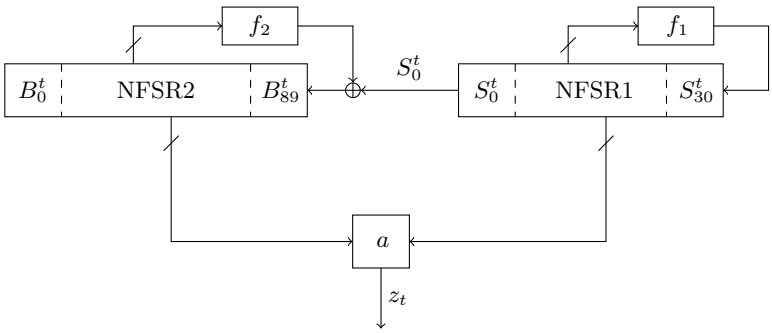

Fig. 1. LIZARD in keystream generation mode.

Note, however, that while Grain uses one linear feedback shift register (LFSR) and one nonlinear feedback shift register (NFSR), which, moreover, are of the same length, LIZARD uses two NFSRs of different lengths instead. The reasons for this design choice will be explained in section 3. Like in Grain, the third important building block besides the two FSRs is a nonlinear output function, which takes inputs from both shift registers and is also used as part of the state initialization algorithm.

In the following, we decribe the components of the cipher in detail (Sec. 2.1) and specify how it is operated during state initialization (Sec. 2.2) and keystream generation (Sec. 2.3). For the sake of clarity, subsections 2.1–2.3 contain only the technical aspects of LIZARD. Explanations of important design choices for our construction are given separately in section 3, along with a discussion of the security properties of the particular components (e.g., the algebraic properties of the feedback functions).

2.1 Components

The 121-bit inner state of LIZARD is distributed over two NFSRs, `NFSR1` and `NFSR2`, whose contents at time $t = 0, 1, \ldots$ we denote by $(S_0^t, \ldots, S_{30}^t)$ and $(B_0^t, \ldots, B_{89}^t)$, respectively (cf. Fig. 1). Note that in LIZARD, `NFSR1` replaces the LFSR of the Grain family of stream ciphers.

NFSR1 is 31 bit wide and corresponds to the NFSR A_{10} of the eSTREAM Phase 2 (hardware portfolio) candidate ACHTERBAHN-128/80 [27]. For non-zero starting states, it has a guaranteed period of $2^{31} - 1$ (i.e., maximal) and is

defined by the following feedback polynomial:

$$\begin{aligned} f_1(x) = 1 &+ x^6 + x^{11} + x^{13} + x^{14} + x^{16} + x^{25} + x^{26} + x^{29} + x^{31} \\ &+ x^9x^{11} + x^{10}x^{14} + x^{10}x^{19} + x^{11}x^{23} + x^{12}x^{17} + x^{13}x^{23} \\ &+ x^9x^{10}x^{11} + x^9x^{11}x^{23} + x^9x^{12}x^{19} + x^9x^{12}x^{27} \\ &+ x^9x^{13}x^{23} + x^9x^{19}x^{27} + x^{10}x^{11}x^{24} + x^9x^{10}x^{11}x^{23} \\ &+ x^9x^{10}x^{12}x^{19} + x^9x^{10}x^{12}x^{27} + x^9x^{10}x^{13}x^{23} \\ &+ x^9x^{10}x^{19}x^{27} + x^{10}x^{11}x^{23}x^{24} + x^{10}x^{12}x^{19}x^{24} \\ &+ x^{10}x^{12}x^{24}x^{27} + x^{10}x^{13}x^{23}x^{24} + x^{10}x^{19}x^{24}x^{27}. \end{aligned}$$

To avoid ambiguity, we also give the corresponding update relations for the individual register stages of `NFSR1`:

$$\begin{aligned} S_i^{t+1} &:= S_{i+1}^t \quad \text{for } i \in \{0, \ldots, 29\}, \\ S_{30}^{t+1} &:= S_0^t \oplus S_2^t \oplus S_5^t \oplus S_6^t \oplus S_{15}^t \oplus S_{17}^t \oplus S_{18}^t \oplus S_{20}^t \oplus S_{25}^t \\ &\quad \oplus S_8^tS_{18}^t \oplus S_8^tS_{20}^t \oplus S_{12}^tS_{21}^t \oplus S_{14}^tS_{19}^t \oplus S_{17}^tS_{21}^t \oplus S_{20}^tS_{22}^t \\ &\quad \oplus S_4^tS_{12}^tS_{22}^t \oplus S_4^tS_{19}^tS_{22}^t \oplus S_7^tS_{20}^tS_{21}^t \oplus S_8^tS_{18}^tS_{22}^t \\ &\quad \oplus S_8^tS_{20}^tS_{22}^t \oplus S_{12}^tS_{19}^tS_{22}^t \oplus S_{20}^tS_{21}^tS_{22}^t \oplus S_4^tS_7^tS_{12}^tS_{21}^t \\ &\quad \oplus S_4^tS_7^tS_{19}^tS_{21}^t \oplus S_4^tS_{12}^tS_{21}^tS_{22}^t \oplus S_4^tS_{19}^tS_{21}^tS_{22}^t \\ &\quad \oplus S_7^tS_8^tS_{18}^tS_{21}^t \oplus S_7^tS_8^tS_{20}^tS_{21}^t \oplus S_7^tS_{12}^tS_{19}^tS_{21}^t \\ &\quad \oplus S_8^tS_{18}^tS_{21}^tS_{22}^t \oplus S_8^tS_{20}^tS_{21}^tS_{22}^t \oplus S_{12}^tS_{19}^tS_{21}^tS_{22}^t. \end{aligned}$$

NFSR2 is 90 bit wide and uses a modified version of g from Grain-128a [48] as its feedback polynomial. More precisely, f_2 squeezes the taps of g as follows to fit a 90 bit register:

$$\begin{aligned} f_2(x) = 1 &+ x^6 + x^{11} + x^{41} + x^{66} + x^{90} + x^{16}x^{30} + x^{31}x^{87} \\ &+ x^{32}x^{35} + x^{37}x^{65} + x^{48}x^{55} + x^{74}x^{75} + x^{78}x^{80} \\ &+ x^{18}x^{22}x^{28} + x^{67}x^{68}x^{70} + x^7x^9x^{10}x^{13}. \end{aligned}$$

Again, to avoid ambiguity, we also give the corresponding update relations for the individual register stages of `NFSR2`:

$$\begin{aligned} B_j^{t+1} &:= B_{j+1}^t \quad \text{for } j \in \{0, \ldots, 88\}, \\ B_{89}^{t+1} &:= S_0^t \oplus B_0^t \oplus B_{24}^t \oplus B_{49}^t \oplus B_{79}^t \oplus B_{84}^t \oplus B_3^tB_{59}^t \oplus B_{10}^tB_{12}^t \\ &\quad \oplus B_{15}^tB_{16}^t \oplus B_{25}^tB_{53}^t \oplus B_{35}^tB_{42}^t \oplus B_{55}^tB_{58}^t \oplus B_{60}^tB_{74}^t \\ &\quad \oplus B_{20}^tB_{22}^tB_{23}^t \oplus B_{62}^tB_{68}^tB_{72}^t \oplus B_{77}^tB_{80}^tB_{81}^tB_{83}^t. \end{aligned}$$

Note that the update relation for B_{89}^{t+1} additionally contains the masking bit S_0^t from `NFSR1` (analogously to the Grain family).

The output function $a : \{0,1\}^{53} \longrightarrow \{0,1\}$ builds on the construction scheme introduced in [43] as part of the FLIP family of stream ciphers (see Sec. 3 for details). For the sake of clarity, we define a through the output bit z_t of LIZARD at time t, which is computed as

$$z_t := \mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t \oplus \tilde{\mathcal{T}}_t,$$

where

$$\mathcal{L}_t := B_7^t \oplus B_{11}^t \oplus B_{30}^t \oplus B_{40}^t \oplus B_{45}^t \oplus B_{54}^t \oplus B_{71}^t,$$

$$\mathcal{Q}_t := B_4^t B_{21}^t \oplus B_9^t B_{52}^t \oplus B_{18}^t B_{37}^t \oplus B_{44}^t B_{76}^t,$$

$$\begin{aligned}\mathcal{T}_t := {} & B_5^t \oplus B_8^t B_{82}^t \oplus B_{34}^t B_{67}^t B_{73}^t \oplus B_2^t B_{28}^t B_{41}^t B_{65}^t \\ & \oplus B_{13}^t B_{29}^t B_{50}^t B_{64}^t B_{75}^t \oplus B_6^t B_{14}^t B_{26}^t B_{32}^t B_{47}^t B_{61}^t \\ & \oplus B_1^t B_{19}^t B_{27}^t B_{43}^t B_{57}^t B_{66}^t B_{78}^t,\end{aligned}$$

$$\tilde{\mathcal{T}}_t := S_{23}^t \oplus S_3^t S_{16}^t \oplus S_9^t S_{13}^t B_{48}^t \oplus S_1^t S_{24}^t B_{38}^t B_{63}^t.$$

2.2 State Initialization

The state initialization process can be divided into 4 phases. Phases 1–3 constitute an instantiation of the $FP(1)$-mode introduced in [32], which, in our case, provides 80-bit security against generic time-memory-data (TMD) tradeoff attacks aiming at key recovery. Phase 4 is a consequence of the necessity to make sure that `NFSR1` is not in the all-zero state after phase 3 (see below for details).[11]

Phase 1: Key and IV Loading. Let $K = (K_0, \ldots, K_{119})$ denote the 120-bit key and $IV = (IV_0, \ldots, IV_{63})$ the 64-bit public IV. The registers of the keystream generator are initialized as follows:

$$B_j^0 := \begin{cases} K_j \oplus IV_j, & \text{for } j \in \{0, \ldots, 63\}, \\ K_j, & \text{for } j \in \{64, \ldots, 89\}, \end{cases}$$

$$S_i^0 := \begin{cases} K_{i+90}, & \text{for } i \in \{0, \ldots, 28\}, \\ K_{119} \oplus 1, & \text{for } i = 29, \\ 1, & \text{for } i = 30. \end{cases}$$

[11] In contrast to Grain, for LIZARD there are no weak key/IV-pairs leading to an all-zero initial state of `NFSR1` after state initialization, which would practically render our maximum-period FSR `NFSR1` ineffective during keystream generation (cf. Sec. 4.10).

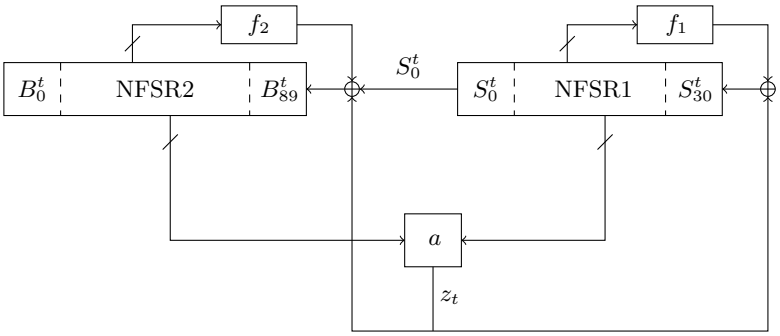

Fig. 2. LIZARD in phase 2 of the state initialization.

Phase 2: Grain-like Mixing. Clock the cipher 128 times without producing actual keystream. Instead, at time $t = 0, \ldots, 127$, the output bit z_t is fed back into both FSRs as depicted in Fig. 2. To avoid ambiguity, we now give the full update relations that will be used for `NFSR1` and `NFSR2` in phase 2. For $t = 0, \ldots, 127$, compute

$$
\begin{aligned}
S_i^{t+1} &:= S_{i+1}^t \quad \text{for } i \in \{0, \ldots, 29\}\,, \\
S_{30}^{t+1} &:= z_t \oplus S_0^t \oplus S_2^t \oplus S_5^t \oplus S_6^t \oplus S_{15}^t \oplus S_{17}^t \oplus S_{18}^t \oplus S_{20}^t \oplus S_{25}^t \\
&\quad \oplus S_8^t S_{18}^t \oplus S_8^t S_{20}^t \oplus S_{12}^t S_{21}^t \oplus S_{14}^t S_{19}^t \oplus S_{17}^t S_{21}^t \oplus S_{20}^t S_{22}^t \\
&\quad \oplus S_4^t S_{12}^t S_{22}^t \oplus S_4^t S_{19}^t S_{22}^t \oplus S_7^t S_{20}^t S_{21}^t \oplus S_8^t S_{18}^t S_{22}^t \\
&\quad \oplus S_8^t S_{20}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{22}^t \oplus S_{20}^t S_{21}^t S_{22}^t \oplus S_4^t S_7^t S_{12}^t S_{21}^t \\
&\quad \oplus S_4^t S_7^t S_{19}^t S_{21}^t \oplus S_4^t S_{12}^t S_{21}^t S_{22}^t \oplus S_4^t S_{19}^t S_{21}^t S_{22}^t \\
&\quad \oplus S_7^t S_8^t S_{18}^t S_{21}^t \oplus S_7^t S_8^t S_{20}^t S_{21}^t \oplus S_7^t S_{12}^t S_{19}^t S_{21}^t \\
&\quad \oplus S_8^t S_{18}^t S_{21}^t S_{22}^t \oplus S_8^t S_{20}^t S_{21}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{21}^t S_{22}^t
\end{aligned}
$$

and

$$
\begin{aligned}
B_j^{t+1} &:= B_{j+1}^t \quad \text{for } j \in \{0, \ldots, 88\}\,, \\
B_{89}^{t+1} &:= z_t \oplus S_0^t \oplus B_0^t \oplus B_{24}^t \oplus B_{49}^t \oplus B_{79}^t \oplus B_{84}^t \oplus B_3^t B_{59}^t \oplus B_{10}^t B_{12}^t \\
&\quad \oplus B_{15}^t B_{16}^t \oplus B_{25}^t B_{53}^t \oplus B_{35}^t B_{42}^t \oplus B_{55}^t B_{58}^t \oplus B_{60}^t B_{74}^t \\
&\quad \oplus B_{20}^t B_{22}^t B_{23}^t \oplus B_{62}^t B_{68}^t B_{72}^t \oplus B_{77}^t B_{80}^t B_{81}^t B_{83}^t,
\end{aligned}
$$

where $z_t := \mathcal{L}_t + \mathcal{Q}_t + \mathcal{T}_t + \tilde{\mathcal{T}}_t$ is computed as described in subsection 2.1.

Phase 3: Second Key Addition. In this phase, the 120-bit key is XORed in a bitwise fashion to the inner state of the keystream generator, similar to how it was introduced in phase 1. More precisely, the following update relations for

NFSR1 and NFSR2 constitute phase 3:

$$B_j^{129} := B_j^{128} \oplus K_j, \quad \text{for } j \in \{0, \ldots, 89\},$$
$$S_i^{129} := \begin{cases} S_i^{128} \oplus K_{i+90}, & \text{for } i \in \{0, \ldots, 29\}, \\ 1, & \text{for } i = 30. \end{cases}$$

Note that, like in phase 1, the rightmost cell of NFSR1 is here again set to 1 in order to avoid the all-zero state, which would practically render our maximum-period FSR NFSR1 ineffective as, after phase 3, the only input to it will come from its feedback function (i.e., for $t \geq 129$, S_i^{t+1}, $i = 0, \ldots, 30$, is computed as described in subsection 2.1). In contrast to phase 1, the key bit K_{119} is not inverted in phase 3.

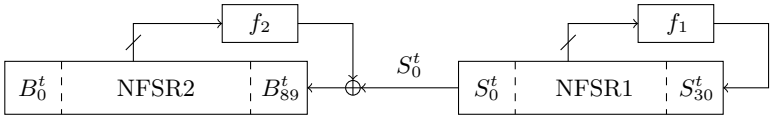

Fig. 3. LIZARD in phase 4 of the state initialization. The output function a is missing as all output bits generated in this phase are discarded.

Phase 4: Final Diffusion. As pointed out previously, phases 1–3 constitute an instantiation of the key loading and mixing steps of the $FP(1)$-mode, which, in our case, yields 80-bit security w.r.t. generic TMD tradeoff attacks aiming at key recovery. Phase 4 is additionally required in order to obtain the necessary diffusion w.r.t. the bit at the rightmost position of NFSR1, which was set to 1 in phase 3. We achieve this by stepping the whole KSG 128 times in keystream generation mode as phase 4 (see Fig. 3). Note that the 128 output bits produced during phase 4 are discarded, i.e., they are not used as actual keystream.

Once more, to avoid ambiguity, we give the full update relations that will be used for NFSR1 and NFSR2 in phase 4. For $t = 129, \ldots, 256$, compute

$$\begin{aligned} S_i^{t+1} &:= S_{i+1}^t \quad \text{for } i \in \{0, \ldots, 29\}, \\ S_{30}^{t+1} &:= S_0^t \oplus S_2^t \oplus S_5^t \oplus S_6^t \oplus S_{15}^t \oplus S_{17}^t \oplus S_{18}^t \oplus S_{20}^t \oplus S_{25}^t \\ &\quad \oplus S_8^t S_{18}^t \oplus S_8^t S_{20}^t \oplus S_{12}^t S_{21}^t \oplus S_{14}^t S_{19}^t \oplus S_{17}^t S_{21}^t \oplus S_{20}^t S_{22}^t \\ &\quad \oplus S_4^t S_{12}^t S_{22}^t \oplus S_4^t S_{19}^t S_{22}^t \oplus S_7^t S_{20}^t S_{21}^t \oplus S_8^t S_{18}^t S_{22}^t \\ &\quad \oplus S_8^t S_{20}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{22}^t \oplus S_{20}^t S_{21}^t S_{22}^t \oplus S_4^t S_7^t S_{12}^t S_{21}^t \\ &\quad \oplus S_4^t S_7^t S_{19}^t S_{21}^t \oplus S_4^t S_{12}^t S_{21}^t S_{22}^t \oplus S_4^t S_{19}^t S_{21}^t S_{22}^t \\ &\quad \oplus S_7^t S_8^t S_{18}^t S_{21}^t \oplus S_7^t S_8^t S_{20}^t S_{21}^t \oplus S_7^t S_{12}^t S_{19}^t S_{21}^t \\ &\quad \oplus S_8^t S_{18}^t S_{21}^t S_{22}^t \oplus S_8^t S_{20}^t S_{21}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{21}^t S_{22}^t, \end{aligned}$$

and

$$
\begin{aligned}
B_j^{t+1} &:= B_{j+1}^t \quad \text{for } j \in \{0, \ldots, 88\}\,, \\
B_{89}^{t+1} &:= S_0^t \oplus B_0^t \oplus B_{24}^t \oplus B_{49}^t \oplus B_{79}^t \oplus B_{84}^t \oplus B_3^t B_{59}^t \oplus B_{10}^t B_{12}^t \\
&\quad \oplus B_{15}^t B_{16}^t \oplus B_{25}^t B_{53}^t \oplus B_{35}^t B_{42}^t \oplus B_{55}^t B_{58}^t \oplus B_{60}^t B_{74}^t \\
&\quad \oplus B_{20}^t B_{22}^t B_{23}^t \oplus B_{62}^t B_{68}^t B_{72}^t \oplus B_{77}^t B_{80}^t B_{81}^t B_{83}^t.
\end{aligned}
$$

See section 3.5 for further remarks on this design choice.

2.3 Keystream Generation

At the end of the state initialization, the 31-bit (initial) state of `NFSR1` is $\left(S_0^{257}, \ldots, S_{30}^{257}\right)$ and the 90-bit (initial) state of `NFSR2` is $\left(B_0^{257}, \ldots, B_{89}^{257}\right)$. The first keystream bit that is used for plaintext encryption is z_{257}. For $t \geq 257$, the states $\left(S_0^{t+1}, \ldots, S_{30}^{t+1}\right)$ and $\left(B_0^{t+1}, \ldots, B_{89}^{t+1}\right)$ and the output bit z_t are computed using the relations given in section 2.1. Fig. 1 (see first paragraph of section 2) depicts the structure of Lizard during keystream generation.

As Lizard is designed to be operated in packet mode, the maximum size of a plaintext packet encrypted under the same key/IV pair is 2^{18} bits and no key/IV pair may be used more than once, i.e., for more than one packet. Let $X = \left(x_0, \ldots, x_{|X|-1}\right)$ denote such a plaintext packet and let $z_{257}, z_{258} \ldots$ be the keystream generated for it as described before. Then the corresponding ciphertext packet $Y = \left(y_0, \ldots, y_{|X|-1}\right)$ can be produced via $y_i := x_i \oplus z_{i+257}$, $i = 0, \ldots, |X| - 1$.[12] Decryption (given that the secret session key and the IV are known) works analogously.

3 Design Considerations

In this section, we provide additional explanations w.r.t. our design, which were omitted in section 2 for the sake of clarity. Based on several of the following properties, we will then argue in section 4, why we believe that Lizard resists the currently known types of attacks against stream ciphers.

3.1 NFSR1

As mentioned in section 2.1, `NFSR1` is 31 bit wide and corresponds to the NFSR A_{10} of the eSTREAM Phase 2 (hardware portfolio) candidate ACHTERBAHN-128/80 [27]. We chose the NFSR A_{10} out of the set of all maximum-period NFSRs

[12] Note that, though we use the terms plaintext/ciphertext packet here, Lizard is really a (synchronous) stream cipher. I.e., the keystream bits $z_{257}, z_{258} \ldots$ are generated in a bitwise fashion (and independently of the plaintext/ciphertext) and, consequently, the individual plaintext bits x_i can be encrypted and then (in the form of y_i) transmitted as they arrive. (The same obviously holds for the decryption of the individual ciphertext bits y_i.)

used in Achterbahn due to the fact that the two NFSRs which would have offered even slightly longer periods (namely, A_{11} and A_{12}) both use inconvenient taps w.r.t. potential speedup measures (see Sec. 3.3 for details). Moreover, we did not choose one of the smaller maximum-period NFSRs of Achterbahn as (under the condition that we want to keep the speedup option as decribed below) those would have led to the situation that we would have had to use certain taps of `NFSR1` at the same time in its feedback function and in the output function. The reason why we used the ACHTERBAHN design as a source for our `NFSR1` in the first place is twofold.

First, we wanted `NFSR1` to have guaranteed period $2^{31} - 1$, as, in LIZARD, `NFSR1` replaces the maximum-length LFSR of the Grain family.[13] Unluckily, apart from special cases like De Bruijn sequences, not much is known yet about how to construct large NFSRs with maximum period.[14] However, due to the restriction to packet mode (with a maximum of 2^{18} keystream bits per key/IV pair), LIZARD actually does not need as large guaranteed periods as the Grain family, which allowed us to replace the maximum-length LFSR of Grain by a suitable maximum-length NFSR. The design document of ACHTERBAHN provides a collection of such maximum-length NFSRs that have sufficient size to be used in LIZARD.

The second reason for using A_{10} from ACHTERBAHN as our `NFSR1` is its hardware efficiency. Despite a comparatively large algebraic normal form, the designers of ACHTERBAHN are able provide a compact hardware realization of the feedback function consuming only 31.75 gate equivalents (GE) and having logical depth three (see Sec. 5 for further details w.r.t. hardware complexity and an explanation of corresponding units of measure like GE).

When operated in a self-contained manner (i.e., after phase 3 of the state initialization), `NFSR1` has a guaranteed period (for non-zero starting states) of $2^{31} - 1$. The following properties of A_{10} of ACHTERBAHN (and, hence, also of `NFSR1` of LIZARD) were given in [27]: a nonlinearity of 61440, an order of correlation immunity of 6, and a diffusion parameter[15] of 61. Moreover, it is

[13] It should be mentioned that in the design document of Grain v1 [34], it is stated that "the LFSR guarantees a minimum period for the keystream and it also provides balancedness in the output." In fact, in the setting of Grain (and also LIZARD), a large guaranteed period of the internal state (composed of both FSRs) is necessary but not sufficient for a large guaranteed period of the keystream, because we only know that the period of the keystream divides the length of the corresponding internal state cycle. The actual guaranteed keystream period of LIZARD (as well as of the Grain family) remains an open problem.

[14] In fact, a De Bruijn sequence of length 2^n can be generated with an n-bit NFSR. Still, for simplicity, we also call our 31-bit `NFSR1` with period $2^{31}-1$ *maximum-length*, in analogy to the case of LFSRs (where a period of 2^n is obviously impossible to achieve with an n-bit LFSR).

[15] The diffusion parameter λ was determined experimentally in [27] and denotes "the minimum number of clock cycles needed in order to transform any two initial states of the shift register A_j of Hamming distance 1 into shift register states of Hamming distance close to $N_j/2$" (N_j denotes the size of the shift register A_j).

easy to see that the feedback function f_1 of NFSR1 is balanced and, thus (as it is 6th order correlation-immune), 6-resilient.[16] Finally, the algebraic degree of f_1 is 4.

3.2 NFSR2

NFSR2 is 90 bit wide and uses a modified version of g from Grain-128a [48] as its feedback polynomial. In contrast to NFSR1, the period of NFSR2 during keystream generation is unknown because, due to the masking bit from NFSR1, NFSR2 is actually a filter instead of a real NFSR (cf. corresponding remark for the Grain family in [33]).

As described in section 2.1, f_2 of Lizard squeezes the taps of g from Grain-128a in such a way that the property of g that no tap appears more than once is preserved in f_2. In consequence, several important properties of g like its balancedness, its resiliency of 4, its nonlinearity of 267403264 and its security w.r.t. linear approximations carry over to f_2 (see [48] for further details regarding the security of g of Grain-128a). The algebraic degree of f_2 is 4.

3.3 Output Function a

An important question in FSR-based stream cipher design is how to share the load of ensuring security between the driving register(s) and the output function. To compensate for the fact that the inner state of Lizard is smaller than that of Grain v1, we decided that the output function should have more inputs and a larger algebraic degree instead. It builds on the construction scheme introduced in [43] as part of the FLIP family of stream ciphers. More precisely, the output function a of Lizard can be written as the direct sum of a linear function with seven monomials, a quadratic function with four monomials, a triangular function with seven monomials, and another triangular function with four monomials, where each tap of NFSR1 and NFSR2 appears at most once in the full output function.

As a consequence, the output function of Lizard is defined over 53 variables, balanced, and has, according to lemmata 3–6 in [43], the following security properties: a nonlinearity of 4476506321453056 ($\approx 2^{51}$), a resiliency of 8, an algebraic immunity of at least 7, and a fast algebraic immunity of at least 8. The algebraic degree of a is 7.

If the content of NFSR1 at time t should be known to the attacker (e.g., as part of a guess-and-determine attack), the output function still depends on at least 43 variables and "gracefully degrades" into the direct sum of a linear function with seven or eight (depending on the values of S_9^t and S_{13}^t) monomials, a quadratic function with four or five (depending on the values of S_1^t and S_{24}^t) monomials, and a triangular function with seven monomials, which again conforms to the construction principle introduced in [43] and leads to the following worst-case

[16] In Grain v1 as well as in Grain-128a, the feedback function of the LFSR (which is replaced by NFSR1 in Lizard) has resiliency 5.

security properties for that situation: a nonlinearity of 4317411672064 ($\approx 2^{41}$), a resiliency of 7, an algebraic immunity of at least 7, and a fast algebraic immunity of at least 8.

While the choice of tap positions for state update functions is often already restricted by the need to guarantee a certain period (e.g., as in the case of `NFSR1`), the choice of tap positions for an output function is commonly less substantiated. For example, the design documents introducing the members of the Grain family (cf. [34], [48], [33]) mainly focus on the conceptual question whether certain taps used in the output function should be from the NFSR or the LFSR (and how many of each). The more concrete question of *which* tap positions within each FSR are actually chosen for the output function is almost exclusively discussed in the context of hardware acceleration or when it comes to mitigate issues of previous versions arising from actual attacks (e.g., the attack of Dinur and Shamir [22] on Grain-128, which lead to a change of tap positions in the output function of Grain-128a [48]).

In the absence of canonical criteria for the selection of tap positions for Grain-like constructions, we mainly resort to the concept of (full) positive difference sets that was used by Golić in [29] to assess the security of nonlinear filter generators consisting of a single LFSR and a nonlinear output function.[17] As it is not clear, however, to what extent these results apply in our case, we move the corresponding discussion to appendix B.

Another cryptanalytic result motivating our selection of tap positions for the output function of Lizard are attacks based on binary decision diagrams (BDDs). A direct consequence of this type of attack against stream ciphers, which was introduced by Krause in [37] and applied to Grain-128 by Stegemann in [52], is that (roughly speaking) the distance between the smallest and the largest tap index of a monomial should be large for as many monomials as possible (see Sec. 4.6 for further details).

3.4 Speedup Options

Our update relation for `NFSR1` does not involve the five taps corresponding to S_{26}^t, S_{27}^t, S_{28}^t, S_{29}^t, S_{30}^t in the generation of S_{30}^{t+1}. Analogously, B_{89}^{t+1} of `NFSR2` is computed without the involvement of B_{85}^t, B_{86}^t, B_{87}^t, B_{88}^t, B_{89}^t. These ten taps are also not used in the output function of the cipher, which allows for a speedup of the keystream generation (up to a factor of 6) simply by multiplying the (comparatively cheap) hardware for the two feedback functions and the output function. Grain-128a uses the same idea to allow for a speedup of up to a factor of 32 (which is obviously not possible here as `NFSR1` has only 31 register cells in total).

3.5 State Initialization Algorithm

As pointed out in section 2.2, the state initialization algorithm of Lizard represents an instantiation of the $FP(1)$-mode introduced in [32], which, in our

[17] A similar approach was taken, e.g., for the NFSR-based stream cipher Espresso [23].

case, provides 80-bit security against generic time-memory-data (TMD) tradeoff attacks aiming at key recovery (cf. Sec. 4.2).

In a nutshell, the generic $FP(1)$-mode for stream ciphers can be described as follows:

- Choose an appropriate[18] keystream generator (KSG) of inner state length n and an appropriate packet length R.
- Define the state initialization algorithm for packet i as follows:
  (1) **Loading:** Given the symmetric session key $k \in \{0,1\}^n$ and the packet initial value $IV^i \in \{0,1\}^n$, load $IV^i \oplus k$ into the inner state registers of the KSG, yielding the inner state $q^i_{load} = IV^i \oplus k$.
  (2) **Mixing:** Run an appropriate KSG-based mixing algorithm[19] on q^i_{load}, yielding the inner state q^i_{mixed}.
  (3) **Hardening:** XOR k bitwise to this inner state q^i_{mixed}, yielding the initial state $q^i_{init} = q^i_{mixed} \oplus k$.
- Starting with the initial state q^i_{init}, the keystream for packet i is generated by the KSG in the traditional way.

The state initialization algorithm of Lizard as described in section 2.2 implements the above scheme with some minor adaptions, which we will describe in the following. Note that none of these modifications reduces the provable 80-bit security against TMD tradeoff attacks provided by the $FP(1)$-mode.

When compared to the generic scheme, the first obvious difference of Lizard is that the IV length is now smaller than the key length. Even in the context of packet mode, we currently do not see the need for IVs larger than 64 bit, hence, this design choice was made to reduce the hardware costs of an implementation.[20] Observe that phase 1 of the state initialization of Lizard (cf. Sec. 2.2) could equivalently be written as

$$B^0_j := \begin{cases} K_j \oplus IV_j, & \text{for } j \in \{0,\ldots,63\}, \\ K_j \oplus 0, & \text{for } j \in \{64,\ldots,89\}, \end{cases}$$

$$S^0_i := \begin{cases} K_{i+90} \oplus 0, & \text{for } i \in \{0,\ldots,28\}, \\ K_{119} \oplus 1, & \text{for } i = 29, \\ 1, & \text{for } i = 30, \end{cases}$$

[18] "Appropriate" means that the keystream generator satisfies standard requirements w.r.t. design goals like security and hardware efficiency. $\frac{2}{3}n$-security against TMD tradeoff attacks is then added by the $FP(1)$-mode.

[19] Such a mixing algorithm can, e.g., simply consist of repeatedly stepping the KSG in keystream generation mode without producing output (cf. Trivium [17]) or involve more sophisticated measures like generating a piece of *internal* keystream that is not output but instead loaded in parallel to the inner state registers of the KSG as the initial state based on which the actual keystream is then generated (cf. the E_0-cipher of the Bluetooth standard [51]).

[20] Note that Grain v1 also uses 64-bit IVs. For extreme situations, where more than 2^{64} packets need to be encrypted, Lizard should be operated in a setting that allows the use of session keys.

which (ignoring S_{30}^0 for now, see below) can be interpreted as the loading phase of the generic $FP(1)$-mode under the restriction that the rightmost 56 bits of the 120-bit IV (in the generic scheme) are fixed to $(0, \ldots, 0, 1)$. In the random oracle model that was used to prove the $\frac{2}{3}n$-security of the $FP(1)$-mode against TMD tradeoff attacks aiming at key recovery (cf. [32] and the citations therein), such a restriction of the choice of IVs in fact means a limitation of an attacker's capabilities. Hence, the security claims for the generic $FP(1)$-mode still hold for Lizard (cf. Sec. 4.2) . The reason for setting $S_{29}^0 := K_{119} \oplus 1$ is to avoid the "sliding property" of Grain v1 and Grain-128 that was pointed out in [20] (see Sec. 4.8 for further details).

Another difference compared to the generic scheme is that in Lizard, the size of the inner state is one bit larger than the size of the key. This extension by one bit is necessary because of the Grain-like, FSR-based structure of Lizard. More precisely, adding a 120-bit key to a 120-bit inner state in phase 3 could result in a situation, where both FSRs get stuck in the all-zero state, leading to an all-zero keystream for the corresponding packet. Extending the inner state to 121 bit and setting $S_{30}^{129} := 1$ prevents this. Again, in the context of the security proof for the $FP(1)$-mode, a TMD tradeoff attack on Lizard (with 120-bit keys and a 121-bit inner state) will have at least the same complexity as an attack on the generic scheme with 120-bit keys and a 120-bit inner state.

The third notable difference between the generic $FP(1)$-mode and the state initialization of Lizard is the additional phase 4 (cf. Sec. 2.2) in Lizard. The purpose of this phase is twofold. First, without it, due to setting $S_{30}^{129} := 1$ in phase 3, one bit of the inner state of `NFSR1` would be known to an attacker during the first 31 steps of keystream generation. Though we do not consider this an immediate threat (remember that the output function of Lizard takes 53 inner state bits as input), it is nonetheless an undesirable property that we feel should be avoided. The second motivation behind phase 4 (and, in fact, also the reason for stepping the whole KSG in phase 4 instead of only stepping `NFSR1`, which would have been sufficient to counter the first problem) is to avoid certain related key/IV properties, which will be discussed in detail as part of section 4.8.

4 Cryptanalysis

Beyond question, the essential feature of a cipher is its security. In particular, new schemes need to resist those attacks which weakened or even broke other ciphers in the past. The term *resist* here refers to a certain security level targeted by the designers, where the common approach is to require that no attack with a complexity lower than that of exhaustive key search exists.[21] For Lizard, we deviate from this *one-security-level-fits-it-all* approach to allow for a hardware efficient

[21] E.g., for Grain v1, the authors state: "The key size is 80 bits and the IV size is specified to be 64 bits. The cipher is designed such that no attack faster than exhaustive key search should be possible, hence the best attack should require a computational complexity not significantly lower than 2^{80}" [34].

implementation in low-cost scenarios, where certain types of attack might be of lesser importance than, e.g., strictly limiting factors like energy consumption. More precisely, LIZARD offers 80-bit security against key recovery and 60-bit security against distinguishing (both via TMD tradoff attacks, cf. Sec. 4.2). Due to the key length of 120 bit, the time complexity of exhaustive key search is even significantly higher than that of key recovery via TMD tradeoff attack (cf. Sec. 4.1).

Note that LIZARD is only one of many cryptographic schemes that deviate from the design paradigm *key length = security level*. Most prominently, in asymmetric cryptography, key sizes significantly larger than the claimed security level are generally accepted. A well-known example from the field of symmetric cryptography is the lightweight authentication protocol by Hopper and Blum [35] (commonly known as HB protocol) with its numerous variants (e.g., the HB$^+$ protocol of Juels and Weis [36]), some of which need key sizes of several hundreds of bits to be operated securely (see [4] for further details).

Trading some security against distinguishing attacks in favor of hardware efficiency seems a plausible option to us as there are many practical scenarios where an attacker will know anyhow that he is dealing with an encrypted data stream. Moreover, the fact that there is a distinguishing attack does not mean that there are necessarily other, more practically relevant attacks of the same complexity as well. Finally, we would like to point out that for prominent lightweight block ciphers like PRESENT [14] and KATAN64 [19], which both have 80-bit key size and 64-bit block size, a complete code book can be built with complexity lower than 2^{80} without considering the ciphers broken.[22] In the same spirit, we consider a TMD tradeoff-based distinguishing attack of time, memory and data complexity 2^{60} against LIZARD acceptable for many use cases.

In the following subsections, we will argue for several types of attacks against stream ciphers (and the Grain family in particular) why we believe that LIZARD will resist them.

4.1 Exhaustive Key Search

Based on experiments[23] using the computer algebra system Magma, the number of possible initial states (under an arbitrarily fixed IV) after phase 4 of the state initialization is expected to be around $2^{119.34}$. Consequently, an attacker who knows a (sufficiently long) piece of keystream of some packet will need at most around $2^{119.34}$ key guesses to find the corresponding initial state, which then allows to generate the complete keystream of this packet. Note that in Trivium and all members of the Grain family, for an arbitrarily fixed IV, the state initialization realizes an injective mapping of the key space to the set of

[22] In [19], also for KATAN32, which has 80-bit key size and 32-bit block size, an 80-bit security level is claimed.

[23] In our experiments, we assume that phase 2 of the state initialization of LIZARD realizes a random, bijective mapping of $\left(B_0^0, \ldots, B_{89}^0, S_0^0, \ldots, S_{30}^0\right)$ to $\left(B_0^{128}, \ldots, B_{89}^{128}, S_0^{128}, \ldots, S_{30}^{128}\right)$.

initial states. In our case, this property is lost due to the second key addition and setting $S_{30}^{129} := 1$ in phase 3 of the state initialization. Still, around $2^{119.34}$ possible initial states for an arbitrarily fixed IV seems more than enough.[24] Moreover, only if the actual key was found by the attacker (and not just one that produces the same initial state for the given IV), he will be able to decrypt other packets as well.

In [20], an attack on Grain v1 and Grain-128 (but not Grain-128a) was introduced that allowed to reduce the cost of exhaustive key search by a factor of two by making use of what the authors call the "sliding property" of those ciphers. In subsection 4.8, we will explain how this sliding property was avoided for LIZARD.

4.2 Time-Memory-Data Tradeoff Attacks

The vulnerability against TMD tradeoff attacks like those of Babbage [6] and Biryukov and Shamir [13] represents an inherent weakness of existing practical stream ciphers. This yields the well-known rule that for achieving l-bit security by a stream cipher (in the standard, non $FP(1)$-type operation mode), one has to choose an inner state length of at least $2l$ for the underlying keystream generator. As a consequence, modern practical stream ciphers have comparatively large inner state lengths (e.g., 288 bit for Trivium [17] or 160 bit for Grain v1 [34]).

Keep in mind that, e.g., for Trivium and all members of the Grain family, not only the state update function during keystream generation but also the state initialization algorithm, which computes the initial state from a key/IV pair, is efficiently invertible. As a consequence, if an attacker manages to recover any inner state during keystream generation, he will also be able to recover the corresponding initial state, and, by inverting the state initialization algorithm, the underlying secret key. While computing the initial state from any of the later inner states is also possible for LIZARD, the secret key cannot be computed efficiently from the initial state.[25] This is due to the fact that LIZARD represents an instantiation of the $FP(1)$-mode as described in section 3.5.

As a consequence, for LIZARD, we have to treat the case of TMD tradoff attacks aiming at key recovery and the case of TMD tradeoff attacks aiming at recovering the initial state of some packet separately. In [32] (building on [38]), it was shown that by using the $FP(1)$-mode, one can reach $\frac{2}{3}n$-bit security against TMD tradeoff attacks aiming at key recovery for a corresponding keystream

[24] Remember that LIZARD is supposed to be a power-saving alternative to Grain v1, whose key space has only size 2^{80}.

[25] Note that even if the state initialization algorithm of conventional stream ciphers like Trivium and Grain (i.e., keystream generators that, after loading key and IV at time $t = 0$, operate without any further external input) would realize a one-way function, one could still launch a TMD tradoff attack in order to recover the inner state at $t = 0$, from which the key could then be derived. This attack would require $\mathcal{O}(2^{n/2})$ sufficiently long keystream pieces generated under different IVs, where n denotes the inner state length of the keystream generator. Due to the second key addition in phase 3, this TMD tradoff attack variant will not work for LIZARD.

generator with inner state length n. In the case of Lizard, this means 80-bit security against key recovery as argued in section 3.5.

Using the classical TMD tradoff attack, an attacker could also try to recover the initial state of some keystream packet generated by Lizard. We will briefly describe the two extreme cases of such an attack, both of which have time, memory and data complexity (at least) 2^{60}. In the first variant, an attacker knows 121 keystream bits of each of approx. 2^{60} keystream packets. A TMD tradoff attack will then allow him to learn one of the corresponding 2^{60} initial states of these keystream packets, based on which he will be able to generate the complete keystream of this packet. Note, however, that the attacker has no control about *which one* of these 2^{60} keystream packets he will eventually be able to generate completely. Moreover, the attacker must know at least 121 bits of the respective keystream packet beforehand, i.e., he will not be able to generate a keystream packet that was not part of his TMD tradeoff attack. In the second extreme case, the attacker tries to keep the set of keystream packets used in his TMD tradeoff attack as small as possible. This is achieved if he knows $121 + 2^{17}$ (i.e., little more than half the maximum packet size of Lizard) keystream bits of each of approx. 2^{43} keystream packets. Then, a TMD tradeoff attack (still with overall time, memory and data complexity 2^{60}) will allow him to learn one of the corresponding 2^{43} initial states of these keystream packets, based on which he will be able to generate the complete keystream of this packet. Again, the attacker has no control about *which one* of these 2^{43} keystream packets he will eventually be able to generate completely and, to launch his attack, he must know more than 2^{17} keystream bits of each of those 2^{43} packets.

As pointed out before, the security against TMD tradeoff-based distinguishing attacks is not increased by implementing the $FP(1)$-mode.[26] Hence, for Lizard, there is a distinguishing attack with time, memory and data complexity 2^{60}.

4.3 Correlation Attacks, Linear Approximations

In [34], the designers of Grain v1 state: "Due to the statistical properties of maximum-length LFSR sequences, the bits in the LFSR are (almost) exactly balanced. This may not be the case for a NFSR when it is driven autonomously. However, as the feedback $g(x)$ is xored with an LFSR-state, the bits in the NFSR are balanced. Moreover, recall that $g(x)$ is a balanced function. Therefore, the bits in the NFSR may be assumed to be uncorrelated to the LFSR bits." The same, in fact, applies to Lizard, where (the maximum-length FSR) `NFSR1` corresponds to Grain's LFSR, `NFSR2` corresponds to Grain's NFSR and $f_2(x)$ corresponds to $g(x)$ (cf. Sec. 2.1 and Sec. 3.1–3.2).

In [43], Méaux et al. point out the importance of "good balancedness, nonlinearity and resiliency properties" of the filtering function in order to withstand

[26] See [32] and the papers cited therein for further details w.r.t. security of the generic $FP(1)$-mode against TMD tradeoff attacks aiming at distinguishing or key recovery.

correlation attacks [50] and fast correlation attacks [44]. As explained in section 3.3, Lizard features a rather heavy output function to compensate for the smaller inner state compared to the original Grain family. It is defined over 53 variables and has nonlinearity 4476506321453056, whereas the output function of Grain v1 is defined over 12 variables and has nonlinearity 1536 and the output function of Grain-128a is defined over 17 variables and has nonlinearity 61440. The resiliency of Lizard's output function is 8 compared to 7 for that of Grain v1 and Grain-128a, respectively.[27]

In [12], Berbain, Gilbert and Maximov present an attack on Grain v0 that combines linear approximations of the NFSR's feedback function and of the output function in order to recover the initial state of the LFSR given a sufficient amount of keystream bits. Once the initial state of the LFSR is known, each keystream bit can be expressed as a linear function in one or two (depending on the LFSR state) bits of the internal bitstream of the NFSR, which allows to efficiently recover the initial state of the NFSR using "a technique which consists of building chains of keystream bits" [12]. Two variants (differing in the LFSR derivation method) of this key recovery attack[28] against Grain v0 are described in [12], the best of which has time complexity 2^{43}, memory complexity 2^{42} and data (i.e., keystream) complexity 2^{38}.

As possible countermeasures, Berbain, Gilbert and Maximov proposed the following modifications [12]: "Introduce several additional masking variables from the NFSR in the keystream bit computation", "replace g by a 2-resilient function", "modify the filtering function h in order to make it more difficult to approximate" and "modify the function g and h to increase the number of inputs". The Grain designers revised their eSTREAM submission accordingly (in particular, seven bits from the NFSR were added linearly to the output function) and suggested Grain v1 [34], for which the authors of [12] acknowledge that "[t]his novel version of Grain appears to be much stronger and is immune against the statistical attacks presented in this paper".

For Grain-128a, the feedback function g of the NFSR was constructed with the above attack in mind. The designers state: "The best linear approximation of g is of considerable interest, and for it to contain many terms, we need the resiliency of the function g to be high. We also need a high nonlinearity in

[27] The resiliency of the output functions of Grain v1 [34] and Grain-128a [48] is not specified explicitly in the respective papers. However, these values can be computed rather easily: For Grain v1, the designers state that the function h, which is part of the full output function, is balanced and correlation immune of the first order. Thus, h is 1-resilient (but not 2-resilient as can be checked easily). By adding seven linear monomials, whose tap positions are disjoint from those used in h, the resiliency of the full output function of Grain v1 increases to 7. (This can be shown using lemmata 3 and 4 from [43].) In Grain-128a, the function h is unbalanced and, hence, has resiliency -1 according to Definition 5 in [43]. By adding eight linear monomials, whose tap positions are disjoint from those used in h, the resiliency of the full output function of Grain-128a increases to 7.

[28] Remember that for the original Grain family, initial state recovery and key recovery are equivalent due to the efficiently invertible state initialization algorithm.

order to obtain a small bias." As a consequence, g was chosen such that it has nonlinearity 267403264 and resiliency 4.

As explained in section 3.2, the feedback function f_2 of NFSR2 in LIZARD squeezes the taps of g of Grain-128a in a way that preserves its balancedness, resiliency and nonlinearity. Moreover, in accordance with the above suggestions from [12] and the construction principle underlying g of Grain-128a (see previous paragraph), the output function of LIZARD has more than three times as many inputs, a much higher nonlinearity and a higher resiliency than those of Grain v1 and Grain-128a (cf. values at the beginning of this subsection) in order to strengthen it against linear approximations.

Note that even if the initial state of NFSR1 of LIZARD (corresponding to the LFSR in the original Grain family) could be recovered by means of linear approximation, the subsequent NFSR initial state recovery procedure for Grain v0 described in section 5 of [12] would not work against NFSR2 of LIZARD. This is due to the fact that the output function of LIZARD will still contain at least ten nonlinear monomials (of degrees $2, \ldots, 7$) over bits from NFSR2 if the content of NFSR1 should be known (cf. Sec. 3.3), making the aforementioned chaining technique from [12] not applicable any more.

4.4 Algebraic Attacks

As pointed out previously, unlike for the members of the original Grain family, the state initialization algorithm of LIZARD is not efficiently invertible. Hence, we would actually have to differentiate here between algebraic attacks aiming at key recovery and algebraic attacks trying to recover the initial state of some keystream packet. However, an attempt to express the observed keystream bits as functions of the 120 key bits and then solve the corresponding system of equations would require to include all state transitions down to $t = 0$. Given that both FSRs are nonlinear and considering the high algebraic degree of the output function (which is used as part of the state update in phase 2 of the state initialization), this is clearly more complex than expressing the observed keystream bits as functions of the 121 bits of the initial state at $t = 257$ and then solving the corresponding system of equations. Consequently, for the remainder of this subsection, we will focus on algebraic attacks that try to recover the initial state of some keystream packet generated by LIZARD.

First of all, note that, to the best of our knowledge, no successful (i.e., having complexity lower than 2^{80} (Grain v1) or 2^{128} (Grain-128a)) algebraic attacks that can recover arbitrary initial states for Grain v1 or Grain-128a have been reported so far.[29] Due to the smaller inner state of LIZARD, the number of variables of the

[29] The currently best result seems to be an algebraic attack by Berbain, Gilbert and Joux against a modified version of Grain-128, which requires 2^{115} computations and 2^{39} keystream bits [11]. They point out, however, that "[t]his attack is not applicable to the original Grain-128". Moreover, note that the required amount of keystream bits (belonging to a single initial state) would not be available for LIZARD due to the maximum packet size of 2^{18} bits.

corresponding system of equations in such an attack would now in fact be lower. This, however, is compensated for by the larger degree of the output function, which is now 7 as compared to 3 for Grain v1 and Grain-128a. As pointed out in section 3.3, LIZARD's output function builds on the construction scheme introduced in [43] and has algebraic immunity of at least 7 and a fast algebraic immunity of at least 8. In addition, now both FSRs are nonlinear and `NFSR1`, which corresponds to the LFSR of the original Grain-family, has algebraic degree 4. Based on these properties, we expect that algebraic attacks against LIZARD will not be efficient (i.e., will not have complexity lower than 2^{80}, which is that of TMD tradeoff-based key recovery as described in subsection 4.2).

4.5 Guess-and-determine Attacks

The output function of LIZARD depends on 53 variables, compared to 12 in Grain v1 and 17 in Grain-128a. As pointed out in in section 3.3, when guessing the shorter `NFSR1`, the output function still depends on at last 53 variables and has the following worst-case security properties: nonlinearity 4317411672064, resiliency 7, algebraic immunity at least 7, and fast algebraic immunity at least 8.[30] Thus an algebraic attack on `NFSR2` similar to the one in [11] will have large enough complexity.

Note that guessing the content of `NFSR1` at $t = 0$ corresponds to guessing the bits $K_{90}, \ldots, K_{119}$ of the secret key. An attack that guesses the content of `NFSR1` at $t = 257$ (i.e., the respective part of the initial state) in order to recover to full initial state will not automatically reveal the secret key because of the use of the $FP(1)$-mode, due to which the state initialization of LIZARD (in constrast to the original Grain family) is not efficiently invertible.[31]

4.6 BDD-based Attacks

In [37], Krause introduced the idea of using binary decision diagrams (BDDs) to attack LFSR-based stream ciphers like A5/1 of the GSM standard or E_0 of Bluetooth [51]. Stegemann later showed in [52], how this approach can be transferred to NFSR-based stream ciphers like Trivum and Grain. In contrast to TMD tradoff attacks or correlation attacks, which potentially require a lot of keystream or ciphertext data, BDD attacks are *short-keystream attacks* in the sense that only the information-theoretic minimum of keystream bits (i.e., often only few more than n bits of keystream for a keystream generator of inner

[30] For comparison, the full output function of Grain v1 (Grain-128a) has nonlinearity 1536 (61440) and algebraic degree 3 (3).

[31] More precisely, when considered individually, phase 2 and phase 4 of the state initialization are obviously efficiently invertible. In particular, the knowledge of $\left(S_0^{257}, \ldots, S_{30}^{257}\right)$ (i.e., the `NFSR1`-related part of the initial state) is already sufficient to efficiently recover $\left(S_0^{129}, \ldots, S_{30}^{129}\right)$ (i.e., the content of `NFSR1` after the second key addition in phase 3). However, when treated as a whole, phases 1–3 (i.e., the state transition from $t = 0$ to $t = 129$) cannot be inverted efficiently.

state length n) is required to recover the corresponding initial state. As pointed out in subsection 4.2, for Trivium and all members of the Grain family, an attack that recovers the initial state is equivalent to an attack that recovers the underlying secret key as the state initialization of these stream ciphers is efficiently invertible. For Lizard, this is not the case due to the use of the $FP(1)$-mode. Hence, for each keystream packet produced by Lizard, a separate BDD attack would be required to recover the corresponding initial state, which would then allow to generate the remaining, unknown keystream bits of that packet.

While we are currently not aware of any BDD attack faster than exhaustive key search against any member of the Grain family, the major design consequence of the BDD-related cryptanalytic results that Stegemann obtained for Grain-like stream ciphers is that the maximum number of what he calls *active monomials* of the feedback functions and the output function should be as large as possible (see [52] for further details). In the setting of Stegemann, for Grain v1, the maximum number of active monomials would be 0 for the LFSR, 6 for the NFSR and 5 for the output function. For Grain-128a, the maximum number of active monomials would be 0 for the LFSR, 3 for the NFSR and 3 for the output function. In comparison, for Lizard, the maximum number of active monomials would be 21 for `NFSR1`, 3 for `NFSR2` and 10 for the output function a. Consequently, we expect that, despite the smaller inner state, Lizard will also perform sufficiently well against BDD attacks.

4.7 Chosen-IV Attacks (Conditional Differentials, Cube Distinguishers)

In [41], Lehmann and Meier studied the security of Grain-128a against dynamic cube attacks and differential attacks. They came to the following conclusion: "To analyse the security of the cipher, we study the monomial structure and use high order differential attacks on both the new and old versions. The comparison of symbolic expressions suggests that Grain-128a is immune against dynamic cube attacks. Additionally, we find that it is also immune against differential attacks as the best attack we could find results in a bias at round 189 out of 256."

Lizard has 128 rounds in phase 2 of the state initialization, where the Grain-like mixing is performed as described in section 2.2 and further explained in section 3.5. On top of that, the key is added again in phase 3 of the state initialization and, finally, the keystream generator is stepped 128 additional times (now in keystream generation mode) before the first keystream bit is output.

Note that the inner state of Lizard (121 bit) is smaller than that of Grain v1 (160 bit) and significantly smaller than that of Grain-128a (256 bit), whereas the output function is more dense. It depends on 53 variables as compared to 12 in Grain v1 and 17 in Grain-128a.[32] The combination of a smaller state and a more dense output function causes a faster diffusion of differentials and of the

[32] The output function of Lizard also has more nonlinear monomials (13) than Grain v1 (8) and Grain-128a (5). Moreover, now both FSRs are nonlinear.

monomial structure for LIZARD. Therefore, we expect that LIZARD is at least as resistant against differential attacks and dynamic cube attacks as Grain v1 and Grain-128a, which seem to be already sufficiently secure in that respect.

4.8 Related Key(/IV) Attacks, Slide Attacks

In [39], Küçük first pointed out a *sliding property* of the state initialization of Grain v1, which was later formally published by De Cannière, Küçük and Preneel in [20] as: "For a fraction of $2^{-2 \cdot n}$ of pairs (K, IV), there exists a related pair (K^*, IV^*) which produces an identical but n-bit shifted key stream." In the same paper, the authors describe how this property can be exploited to speed up exhaustive key search for Grain v1 (and also for Grain-128) by a factor of two.[33] As a reaction, the designers of Grain-128a changed the 22-bit constant $(1, \ldots, 1)$ that was used in state initialization of Grain-128 to $(1, \ldots, 1, 0)$.

For LIZARD, a speed-up of exhaustive key search by a factor of two would actually be tolerable due to the key length of 120 bit together with the fact that we aim for 80-bit security against key recovery. Still, we consider such a sliding property undesirable as it might pave the way for other attacks. To avoid it, we set $S_{29}^0 := K_{119} \oplus 1$ in phase 1 of the state initialization (cf. Sec. 3.5). As a result, for a key/IV pair (K, IV), a related key/IV pair (K^*, IV^*) in the sense of [20] would have to satisfy $K_{118}^* = K_{119} \oplus 1$. This, however, would then lead to a bad (i.e., inverted) key bit being added in phase 3 of the state initialization, where $K_{118}^* = K_{119}$ must hold for the attack to work. Note that, without inverting K_{119} in phase 1 of the state initialization, LIZARD would in fact suffer from a variant of the sliding property, despite the second key addition phase 3.

Another undesirable related key/IV property, which would arise out of using the $FP(1)$-mode with a Grain-like keystream generator, is avoided by phase 4 of the state initialization of LIZARD. More precisely, given some key/IV pair (K, IV) together with the corresponding keystream packet, an attacker might know (or suspect) for some other keystream packet that it was generated under the related key/IV pair (K', IV'), where $K_2' = K_2 \oplus 1$, $IV_2' = IV_2 \oplus 1$, $K_i' = K_i$ for $0 \leq i \leq 79$, $i \neq 2$, and $IV_i' = IV_i$ for $0 \leq i \leq 63$, $i \neq 2$. Without phase 4 (i.e., if z_{129} instead of z_{257} was the first keystream bit used for plaintext encryption), the attacker could immediately conclude that the first 11 keystream bits generated for (K', IV') are identical to the first 11 keystream bits $z_{129}, \ldots, z_{139}$ generated for (K, IV). This is due to the fact that if $K_i \oplus IV_i = K_i' \oplus IV_i'$ for $i = 0, \ldots, 63$ and $K_i = K_i'$ for $i = 64, \ldots, 119$, then the corresponding inner states of LIZARD will be identical after phase 2 of the state initialization. In the above example, where key bit K_2 and IV bit IV_2 are flipped for K_2' and IV_2', respectively, this would lead to identical inner states after the second key addition at $t = 129$ except the bit B_2^{129} of `NFSR2`, which would be flipped now as well. However, it takes ten further steps of the keystream generator for this information to reach

[33] In addition, they also suggest a related-key slide attack, for which they note: "As is the case for all related key attacks, the simple attack just described is admittedly based on a rather strong supposition." [20]

a tap of the output function. While the described scenario might seem rather far-fetched, we still wanted to avoid this kind of nonrandom behavior in order to thwart other attacks that might make us of it. Moreover, as pointed out in section 3.5, phase 4 also became necessary due to setting $S_{30}^{129} := 1$ in phase 3.

4.9 IV Collisions

A consequence of setting $S_{30}^{129} := 1$ in phase 3 of the state initialization of Lizard (cf. Sec. 2.2) is that what we call *IV collisions* can occur, i.e., two key/IV pairs (K, IV) and (K, IV') with $IV \neq IV'$ can map to the same initial state and, hence, result in identical keystream packets.[34] The implications of this are twofold.

First, an attacker could try to exploit this fact to launch a distinguishing attack. More precisely, if the corresponding oracle answers with identical, sufficiently long keystream packets for two different key/IV pairs (K, IV) and (K, IV'), where $IV \neq IV'$, then the attacker can distinguish the pseudo-random from the random the scenario. However, as proved in appendix C, the number of such oracle queries necessary for finding a collision with probability $> 1/2$ exceeds 2^{60}, i.e., the complexity of this distinguishing attack is not lower than that of the generic TMD tradeoff distinguishing attack mentioned in section 4.2.

Second, an attacker might get hold of a (partial) keystream packet x generated for the key/IV pair (K, IV) and a partial keystream packet y generated for (K, IV'), where $IV \neq IV'$. If $|x| > |y| > 121$ and y fully coincides with the corresponding part of x, then the attacker can conclude that, with high probability, both (partial) keystream packets x and y where generated based on the same initial state. Consequently, he would now know a larger piece (i.e., x) of the keystream packet for (K, IV'). However, the expected number of (partial) keystream packets needed for finding such a collision is larger than 2^{60} and the attacker would have no choice w.r.t. *for which one* of these packets he would obtain further keystream bits. Overall, this scenario has the same complexity as the TMD tradoff-based initial state recovery attack described in section 4.2 and seems even less realistic.

4.10 Weak Key/IV Pairs

In [54], Zhang and Wang introduced the notion of *weak key/IV pairs* for the Grain family of stream ciphers. They use such pairs, which lead to an all-zero initial state of the LFSR, to mount distinguishing attacks and initial state recovery attacks.[35] In [48], the designers of Grain-128a point out: "We note that the IV is normally assumed to be public, and that the probability of using a weak key/IV pair is 2^{-128}. Any attacker guessing this to happen and then launching a rather expensive attack, is much better off just guessing a key." For Grain

[34] For members of the original Grain family, such IV collisions are not possible.

[35] Keep in mind that, for all members of the original Grain family, initial state recovery is equivalent to key recovery as pointed out in section 4.2.

v1, which has 2^{64} weak key/IV pairs among a total of 2^{144} key/IV pairs, the corresponding probability would be 2^{-80}, leading to a similar conclusion.

In analogy to the definition of Zhang and Wang, weak key/IV pairs for Lizard would lead to an all-zero initial state of `NFSR1`. This, however, is impossible due to setting $S_{30}^{129} := 1$ in phase 3 of the state initialization (cf. Sec. 2.2) togther with the fact that, after phase 3, the only input to the maximum-period FSR `NFSR1` comes from its own feedback function.

Note that, without setting $S_{30}^{129} := 1$ in phase 3 of the state initialization, there would have been about 2^{153} weak key/IV pairs out of 2^{184} total key/IV pairs for Lizard, leading to a probability of using a weak key/IV of 2^{-31}. Consequently, without setting $S_{30}^{129} := 1$ in phase 3 of the state initialization, attacks based on weak key/IV pairs might have posed a real threat to Lizard.

5 Hardware Results

In 2004, the eSTREAM project was started in order to identify new stream ciphers for two application profiles: "Profile 1 contains stream ciphers more suitable for software applications with high throughput requirements. Profile 2 stream ciphers are particularly suitable for hardware applications with restricted resources such as limited storage, gate count, or power consumption." [45]

The competition ended in 2008 and, for profile 2 (hardware), the resulting eSTREAM portfolio contained four ciphers, one of which was removed shortly after due to new cryptanalytic results. After the latest review in 2012 [7], the remaining three ciphers for profile 2 are still part of the portfolio: Grain v1, MICKEY 2.0 and Trivium.

In this section, we present the hardware results for our new stream cipher Lizard and compare them to those of Grain v1. The reasons for focusing on Grain v1 are twofold. First, it is a natural choice for comparison due to the close structural relation between Lizard and Grain v1 as explained in sections 2 and 3. Second, and more importantly, Grain v1 turned out to be the most hardware efficient member of the eSTREAM portfolio and, hence, can be considered as a benchmark for new designs.[36]

When comparing the hardware performance of ciphers, the first apparent task is to specify which hardware is actually targeted. In the spirit of the aforementioned requirements for profile 2 of the eESTREAM contest and in line with corresponding papers like [25] and [30], we focus on application-specific integrated circuits (ASICs) with standard CMOS libraries.[37] ASICs are an essential component in radio frequency identification (RFID) technology, which,

[36] This conclusion can be drawn from tables 1–4 and figures 1–3 in [30], where Good and Benaissa evaluate the hardware performance of the phase-3, profile-2 candidates of the eSTREAM competition. Note that, for the sake of comparability, we are referring to the standard, non-parallelized implementations of the ciphers, which (after initialization) produce one keystream bit per clock cycle.

[37] The testing framework for eSTREAM profile-2 candidates [10] additionally considers low-cost FPGAs and a corresponding performance evaluation can be found in, e.g., [31] and [16]. However, as ASICs are certainly more common in ultra-constrained,

as Feldhofer puts it, "allows giving a digital identity to nearly every object in the world" [25]. While especially low-cost RFID tags may be subject to severe resource limits (see, e.g., [4]), in many cases, they still need to provide security features like confidentiality or privacy. The two main restrictions imposed on the design of cryptographic protocols for RFID tags are the circuit size and the power budget. The circuit size strongly influences the manufacturing costs of an RFID tag (see [4] for details) and is commonly specified in gate eqivalents (GE), where one GE corresponds to the area of a two-input drive-strength-one NAND gate. The power consumption is crucial as low-cost RFID tags are usually passively powered (i.e., via an electromagnetic field radiated by the reader). Given that the transmission power of an RFID reader is limited by factors like legal regulations, the more power a tag consumes, the smaller the maximum reading distance becomes. As done by Feldhofer in [25] for his comparison of low-power implementations of Trivium and Grain, we will focus on these two values, cell area and power consumption, in our comparison of Lizard and Grain v1. In addition, we provide the delay[38] of the respective circuits and the number of clock cycles required to perform the state initialization in Tab. 1. Like Feldhofer, we will not provide compound metrics (e.g., the power-area-time product given in [30]) but leave the computation (as appropriate for the respective application scenario) to the reader.

It is important to note that while the area requirement of cipher designs can be compared over different standard cell libraries by using the measure gate equivalents, "[p]ower cannot be scaled reliably between different processes and libraries" [30]. Consequently, it is inevitable to use the same design flow for all implementations that are to be compared. As done by Good and Benaissa in [31] and [30] in their hardware comparison of eSTREAM candidates, we use Cadence tools for synthesis and ModelSim for generating switching activity. While Feldhofer uses 0.35 μm and Good and Benaissa use 0.13 μm CMOS process technology in the cited papers, we employ the UMCL18G212T3 (0.18 μm,

low-cost environments (targeted by Lizard), we focus on this technology and leave an FPGA-related evaluation of our new cipher as future work.

[38] The critical path delay of the circuit determines the maximum clock frequency and, hence, the maximum achievable throughput. However, for low-cost RFIDs, 100 kHz is the commonly assumed clock frequency (see, e.g., [4], [25], [30]), which would allow a delay of up to 10^7 ps. For comparison, the delay of our implementation of Lizard is 2474 ps, which corresponds to a maximum clock frequency of about 404 MHz. Also note that, though targeting low-cost devices, Lizard can still be used in scenarios where throughputs larger than 404 Mbit/s are required, simply by using techniques like pipelining (as done for the stream cipher Espresso in [23]) in order to reduce the delay or by implementing a parallelized version of Lizard as described in Sec. 3.4. Moreover, it is possible to instruct the synthesis tool to optimize for higher clock speeds, which will lead to a circuit with smaller delay but, inter alia, higher area requirements. However, to keep the comparison between Lizard and Grain v1 as concise as possible, we only give the delays for the design flow targeting 100 kHz clocks as these values already show sufficiently well that Lizard is also suitable for higher speed applications.

1.8 V) standard cell library that was also used by Poschmann in [47] for implementing the block cipher PRESENT.[39] Our results (see Tab. 1) are obtained via Encounter RTL Compiler RC12.22 and are based on the netlist generated through the command `synthesize -to_placed -effort high`. Like Feldhofer in [25], we target a 100 kHz clock and employ clock gating. The switching activity for power estimation (recorded with ModelSim SE-64 6.5b and fed back to RTL compiler) covers the generation of 10 kbit of keystream (as done in [31]) at a clock rate of 100 kHz and includes the state initialization of the compared cipher modules. To improve the accuracy of the results, switching activity for 25 different random key/IV combinations is considered and the arithmetic mean of the respective power estimations is computed.[40]

As pointed out above, we had to implement not only our new cipher Lizard but also Grain v1 in order to obtain a meaningful comparison of power consumptions on the basis of the same design flow. Moreover, as Lizard targets low-power environments, we decided to serialize phases 1 (i.e., the initial key and IV loading) and 3 (i.e., the second key addition) of its state initialization (see appendix D.1 for details). This allows to take full advantage of the FSR-based structure of the keystream generator by using simple D flip-flops (without additional costly features like scan, set/reset or enable functionality) for storing the cipher's inner state. Naturally, for reasons of fairness, we considered the positive effects of serializing the key/IV loading for Grain v1 as well. The corresponding values can be found in Tab. 1 along with those for the straightforward implementation of Grain v1. We do not suggest to use non-serialized key/IV loading for Lizard and, hence, Tab. 1 only contains the hardware metrics for Lizard with phases 1 and 3 implemented as described in appendix D.1.[41]

Serializing parts of the state initialization comes at a price, however, as it increases latency. In the case of Lizard, 240 additional clock cycles are required for computing the initial state, based on which, subsequently, one keystream bit per clock cycle is produced. We consider this increase tolerable considering that, even with serialized key/IV loading, the state initialization (including module reset) of Lizard takes only 499 clock cycles, as compared to, e.g., 1153 clock cycles for the straightforward (i.e., non-serialized) implementation of the state initialization of the eSTREAM portfolio member Trivium. For Grain v1, serializing the key/IV loading can be realized at the cost of only 80 additional clock

[39] The main reason for choosing UMCL18G212T3 is that we have already used this library in various projects and, hence, have most experience with it.

[40] For all power values given in Tab. 1, the largest deviation of a single estimation from the computed average was below one percent.

[41] The reason for not describing phases 1 and 3 of the state initialization of Lizard in its serialized form in the first place in Sec. 2.2 is that we wanted the specification of the algorithm to be as concise as possible. Moreover, the way we introduce Lizard in Sec. 2.2 hopefully facilitates to understand the relation to the $FP(1)$-mode as described in Sec. 3.5.

Design	Area [GE]	Power [nW]	Delay [ps]	Latency [clk. cyc.]
Lizard*	1161	2110	2474	499
Grain v1*	1268	2517	2155	241
Grain v1	1221	3578	2166	161

Table 1. Hardware results for a clock speed of 100 kHz. The symbol * indicates that the respective implementation uses serialized key/IV loading. Latency represents the number of clock cycles needed to perform the state initialization. After state initialization, all designs produce one keystream bit per clock cycle, corresponding to a throughput of 100 Kbit/s.

cycles as the key and the IV can be shifted separately into the NFSR and the LFSR, respectively, at the same time.[42]

Tab. 1 shows that the estimated power consumption of Lizard during the generation of 10 kbit of keystream (including state initialization) is about 16 percent lower than that of Grain v1 with serialized key/IV loading. Moreover, Lizard also allows to save on chip area and, hence, production costs.

At first glance, the reduction in chip area might seem surprisingly small, considering that Lizard's inner state is about 25 percent smaller than that of Grain v1. Remember, however, that Lizard needs additional logic for loading the (larger) key (twice) and, moreover, we chose a much heavier output function, which is defined over 53 variables.

As a consequence of the larger key, Lizard would benefit more than Grain v1 from an "external" key source (like an EEPROM) that takes over the task of key bit selection based on an index or supplies the key bits sequentially. However, for reasons of fairness, we assumed the (from Lizard's point of view) worst situation that all key and IV bits are provided via separate wires to the respective cipher modules, which then have to take care of key bit selection themselves. To avoid ambiguity about the capabilities of the implementations which the values in Tab. 1 are based on, we provide the interfaces of the respective modules along with a short description in appendix D.2.

6 Conclusion

We presented Lizard, a new lightweight stream cipher for power-constrained devices like passive RFID tags. Its hardware efficiency results from combining a

[42] In our non-serialized (i.e., straightforward) implementation of Grain v1, the key/IV loading is performed as part of the module reset (which takes one clock cycle). Hence, the state initialization takes $1 + 160 = 161$ clock cycles. In the serialized implementation, the key/IV loading of Grain v1 is performed in a separate stage, which spans 80 clock cycles. Hence, the state initialization takes $1 + 80 + 160 = 241$ clock cycles.

Grain-like design with the $FP(1)$-mode, a recently suggested construction principle for the state initialization of stream ciphers, which offers provable $\frac{2}{3}n$-security against TMD tradeoff attacks aiming at key recovery. LIZARD uses 120-bit keys, 64-bit IVs and has an inner state length of 121 bit. It is supposed to provide 80-bit security against key recovery attacks and 60-bit security against distinguishing attacks. LIZARD allows to generate up to 2^{18} keystream bits per key/IV pair, which would be sufficient for many existing communication scenarios like Bluetooth, WLAN or HTTPS.

Hardware implementations for LIZARD and Grain v1 were created using the same design flow in order to allow for a meaningful comparison of performance metrics. The results show that LIZARD consumes about 16 percent less power than Grain v1 at slightly reduced area requirements. This indicates that in scenarios where plaintext packets of moderate length are to be encrypted separately under individual IVs, the $FP(1)$-mode provides an interesting alternative to conventional state initialization algorithms of stream ciphers.

As future work, we suggest to evaluate the performance of LIZARD on other hardware platforms like FPGAs or microcontrollers. Moreover, it might be interesting to investigate, whether, under the current security guarantees, even more lightweight variants of LIZARD are possible (e.g., by choosing a less heavy output function). With respect to the generic $FP(1)$-mode, it would be desirable to have key sizes of $\frac{2}{3}n$ (instead of currently n) and still maintain provable $\frac{2}{3}n$-security against TMD tradeoff attacks aiming at key recovery. One way to achieve this could be to derive an n-bit $FP(1)$-*mode key* (or even two separate n-bit keys for the respective phases of the state initialization) on-the-fly from the actual $\frac{2}{3}n$-bit key, e.g., by using a component like the round key function of the stream cipher Fruit.

Acknowledgement

We would like to thank Peter Fischer and Michael Ritzert, who provided us with the necessary technical means and additional valuable information for creating the hardware implementation of LIZARD.

References

1. IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific Requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-1997*, pages i–445, 1997.
2. IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific Requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements. *IEEE Std 802.11i-2004*, pages 1–190, July 2004.

3. IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific Requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793, March 2012.
4. Frederik Armknecht, Matthias Hamann, and Vasily Mikhalev. *Lightweight Authentication Protocols on Ultra-Constrained RFIDs - Myths and Facts*, pages 1–18. Springer International Publishing, Cham, 2014.
5. Frederik Armknecht and Vasily Mikhalev. *On Lightweight Stream Ciphers with Shorter Internal States*, pages 451–470. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
6. S.H. Babbage. Improved "exhaustive search" attacks on stream ciphers. In *Security and Detection, 1995., European Convention on*, pages 161–166, May 1995.
7. Steve Babbage, Julia Borghoff, and Vesselin Velichkov. D.SYM.10 - the eSTREAM portfolio in 2012. eSTREAM: the ECRYPT Stream Cipher Project, 2012. `http://www.ecrypt.eu.org/ecrypt2/documents/D.SYM.10-v1.pdf`.
8. Steve Babbage and Matthew Dodd. The stream cipher MICKEY 2.0 (eSTREAM). Technical report, ECRYPT (European Network of Excellence for Cryptology), 2006. `http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf`.
9. Elad Barkan and Eli Biham. *Conditional Estimators: An Effective Attack on A5/1*, pages 1–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
10. Lejla Batina, Sandeep Kumar, Joseph Lano, Nele Lemke, Kirstin Mentens, Christof Paar, Bart Preneel, Kazuo Sakiyama, and Ingrid Verbauwhede. Testing framework for eSTREAM Profile II candidates. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/014, 2006. `http://www.ecrypt.eu.org/stream/papersdir/2006/014.pdf`.
11. Côme Berbain, Henri Gilbert, and Antoine Joux. *Algebraic and Correlation Attacks against Linearly Filtered Non Linear Feedback Shift Registers*, pages 184–198. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
12. Côme Berbain, Henri Gilbert, and Alexander Maximov. *Fast Software Encryption: 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, chapter Cryptanalysis of Grain, pages 15–29. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
13. Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer Berlin Heidelberg, 2000.
14. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. *PRESENT: An Ultra-Lightweight Block Cipher*, pages 450–466. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
15. Marc Briceno, Ian Goldberg, and David Wagner. A pedagogical implementation of A5/1, 1999. Available at `http://www.scard.org/gsm/a51.html`.
16. Philippe Bulens, Kassem Kalach, Francois-Xavier Standaert, and Jean-Jacques Quisquater. FPGA implementations of eSTREAM phase-2 focus candidates with hardware profile. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/024, 2007. `http://www.ecrypt.eu.org/stream/papersdir/2007/024.pdf`.
17. Christophe De Cannière and Bart Preneel. Trivium - specifications (eSTREAM). Technical report, ECRYPT (European Network of Excellence for Cryptology), 2005. `http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf`.

18. P. H. Cole and D. C. Ranasinghe. *Networked RFID Systems and Lightweight Cryptography: Raising Barriers to Product Counterfeiting*. Springer Berlin Heidelberg, 1 edition, 2008.
19. Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. *KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers*, pages 272–288. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
20. Christophe De Cannière, Özgül Küçük, and Bart Preneel. *Analysis of Grain's Initialization Algorithm*, pages 276–289. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
21. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919.
22. Itai Dinur and Adi Shamir. *Breaking Grain-128 with Dynamic Cube Attacks*, pages 167–187. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
23. Elena Dubrova and Martin Hell. Espresso: A stream cipher for 5g wireless communication systems. Cryptology ePrint Archive, Report 2015/241, 2015. `http://eprint.iacr.org/`.
24. Muhammed F. Esgin and Orhun Kara. *Practical Cryptanalysis of Full Sprout with TMD Tradeoff Attacks*, pages 67–85. Springer International Publishing, Cham, 2016.
25. Martin Feldhofer. Comparison of low-power implementations of Trivium and Grain. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/027, 2007. `http://www.ecrypt.eu.org/stream/papersdir/2007/027.pdf`.
26. Scott Fluhrer, Itsik Mantin, and Adi Shamir. *Weaknesses in the Key Scheduling Algorithm of RC4*, pages 1–24. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.
27. B. M. Gammel, Rainer Göttfert, and Oliver Kniffler. Achterbahn-128/80. eSTREAM: the ECRYPT Stream Cipher Project, 2006. `http://www.ecrypt.eu.org/stream/p2ciphers/achterbahn/achterbahn_p2.pdf`.
28. Vahid Amin Ghafari, Honggang Hu, and Chengxin Xie. Fruit: Ultra-lightweight stream cipher with shorter internal state. Cryptology ePrint Archive, Report 2016/355, 2016. `http://eprint.iacr.org/2016/355`.
29. Jovan Dj. Golić. *Fast Software Encryption: Third International Workshop Cambridge, UK, February 21–23 1996 Proceedings*, chapter On the security of nonlinear filter generators, pages 173–188. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.
30. Tim Good and Mohammed Benaissa. Hardware performance of eStream phase-III stream cipher candidates. eSTREAM: the ECRYPT Stream Cipher Project, 2008. `http://www.ecrypt.eu.org/stream/docs/hardware.pdf`.
31. Tim Good, William Chelton, and Mohamed Benaissa. Review of stream cipher candidates from a low resource hardware perspective. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/016, 2006. `http://www.ecrypt.eu.org/stream/papersdir/2006/016.pdf`.
32. Matthias Hamann and Matthias Krause. Stream cipher operation modes with improved security against generic collision attacks. Cryptology ePrint Archive, Report 2015/757, 2015. `http://eprint.iacr.org/2015/757`.
33. Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. *New Stream Cipher Designs: The eSTREAM Finalists*, chapter The Grain Family of Stream Ciphers, pages 179–190. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

34. Martin Hell, Thomas Johansson, and Willi Meier. Grain - a stream cipher for constrained environments. eSTREAM: the ECRYPT Stream Cipher Project, 2006. http://www.ecrypt.eu.org/stream/p3ciphers/grain/Grain_p3.pdf.
35. Nicholas J. Hopper and Manuel Blum. *Secure Human Identification Protocols*, pages 52–66. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.
36. Ari Juels and Stephen A. Weis. *Authenticating Pervasive Devices with Human Protocols*, pages 293–308. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
37. Matthias Krause. *BDD-Based Cryptanalysis of Keystream Generators*, pages 222–237. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.
38. Matthias Krause. Analyzing constructions for key-alternating pseudorandom functions with applications to stream cipher operation modes. Cryptology ePrint Archive, Report 2015/636, 2015. http://eprint.iacr.org/2015/636.
39. Özgül Küçük. Slide resynchronization attack on the initialization of grain 1.0. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/044, 2006. http://www.ecrypt.eu.org/stream.
40. Virginie Lallemand and María Naya-Plasencia. *Cryptanalysis of Full Sprout*, pages 663–682. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
41. Michael Lehmann and Willi Meier. *Conditional Differential Cryptanalysis of Grain-128a*, pages 1–11. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
42. Yi Lu, Willi Meier, and Serge Vaudenay. *The Conditional Correlation Attack: A Practical Attack on Bluetooth Encryption*, pages 97–117. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
43. Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. Cryptology ePrint Archive, Report 2016/254, 2016. http://eprint.iacr.org/.
44. Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989.
45. ECRYPT European Network of Excellence for Cryptology. eSTREAM: the ECRYPT stream cipher project, 2008. http://www.ecrypt.eu.org/stream/.
46. A. Popov. Prohibiting RC4 Cipher Suites. RFC 7465 (Proposed Standard), February 2015.
47. Axel Poschmann. Lightweight cryptography - cryptographic engineering for a pervasive world. Cryptology ePrint Archive, Report 2009/516, 2009. http://eprint.iacr.org/2009/516.
48. Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: A new version of Grain-128 with optional authentication. *Int. J. Wire. Mob. Comput.*, 5(1):48–59, December 2011.
49. Bruce Schneier. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.
50. T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Trans. Comput.*, 34(1):81–85, January 1985.
51. Bluetooth SIG. Bluetooth core specification 4.2, 2014. https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=286439.
52. Dirk Stegemann. *Extended BDD-Based Cryptanalysis of Keystream Generators*, pages 17–35. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
53. Bin Zhang and Xinxin Gong. *Another Tradeoff Attack on Sprout-Like Stream Ciphers*, pages 561–585. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
54. Haina Zhang and Xiaoyun Wang. Cryptanalysis of stream cipher grain family. Cryptology ePrint Archive, Report 2009/109, 2009. http://eprint.iacr.org/.

A Test Vectors

Key (120 bit), IV (64 bit) and the corresponding first 128 keystream bits in hexadecimal notation. To avoid ambiguity, note that, e.g., the key

`0x01234FFFFFFFFFFFFFFFFFFFFFFF`

corresponds to

$$(K_0, \ldots, K_{119}) = (0,0,0,0,0,0,0,1,0,0,1,0,0,0,1,1,0,1,0,0,1,\ldots,1)\,.$$

Similarly, for the keystream, the example

`0x01000000000000000000000000000000`

would mean that the first seven keystream bits (i.e., $z_{257}, \ldots, z_{263}$) are zero, followed by a one and 120 more zeros.

```
Key:       0x00000000000000000000000000000000
IV:        0x0000000000000000
Keystream: 0xE2FB09F0D7467CC4998768E7FB2CB83C

Key:       0x00000000000000000FFFFFFFFFFFFFFF
IV:        0xFFFFFFFFFFFFFFFF
Keystream: 0x5A3663603CF477F4E11214DA858C10FC

Key:       0x0123456789ABCDEF0123456789ABCD
IV:        0xABCDEF0123456789
Keystream: 0x9D8D1B31CDEFBD32F7D3069AF6CE74B5
```

B Output Function a: Tap Selection

As pointed out in section 3.3, due to the absence of canonical criteria for the selection of tap positions for Grain-like constructions, we mainly resort to the concept of (full) positive difference sets that was used by Golić in [29] to assess the security of nonlinear filter generators consisting of a single LFSR and a nonlinear output function: "for a positive integer λ, call Γ a λth-order positive difference set if λ is the maximum number of pairs of its elements with the same mutual difference (for $\lambda = 1$, we get a full positive difference set)" [29].

In particular, the output function a of Lizard has the following properties:

– The set

$$\{1, 3, 9, 13, 16, 23, 24\}$$

of output function taps from `NFSR1` is a 2nd-order positive difference set. Moreover, no taps from `NFSR1` are used at the same time for its feedback function and the output function.[43]

[43] In Grain-128a, the feedback function of the LFSR (corresponding to `NFSR1` in our construction) and the output function do not share any taps, either.

- No taps from NFSR2 are used at the same time for its feedback function and the output function.[44]
- The direct sum $\mathcal{L}_t + \mathcal{Q}_t + \mathcal{T}_t$ uses only taps from NFSR2. (To maintain a sufficient security level even when the content of the smaller NFSR1 is known to the attacker, e.g., due to guessing; cf. Sec. 4.5.)
- The set
$$\{5, 7, 11, 30, 40, 45, 54, 71\}$$
of the tap indices (all from NFSR2) of the linear monomials of $\mathcal{L}_t + \mathcal{Q}_t + \mathcal{T}_t$ is a full positive difference set.
- The set
$$\{4, 8, 9, 18, 21, 37, 44, 52, 76, 82\}$$
of the tap indices (all from NFSR2) of the quadratic monomials of $\mathcal{L}_t + \mathcal{Q}_t + \mathcal{T}_t$ is a full positive difference set. One consequence of this is that each two bits of the internal bitstream of NFSR2 can form at most once a quadratic monomial together.
- The sets
$$\{|4 - 21|, |8 - 82|, |9 - 52|, |18 - 37|, |44 - 76|\}$$
of differences between the two taps (all from NFSR2) of each quadratic monomial in $\mathcal{L}_t + \mathcal{Q}_t + \mathcal{T}_t$ and
$$\{|3 - 59|, |10 - 12|, |15 - 16|, |25 - 53|, |35 - 42|\}$$
of differences between the two taps (all from NFSR2) of each quadratic monomial in the feedback function of NFSR2 are disjoint. Hence, even during phase 2 of the state initialization, each two bits of the internal bitstream of NFSR2 can form at most once a quadratic monomial together.
- None of the differences
$$\{|4 - 21|, |8 - 82|, |9 - 52|, |18 - 37|, |44 - 76|\}$$
between the two taps (all from NFSR2) of each quadratic monomial in $\mathcal{L}_t + \mathcal{Q}_t + \mathcal{T}_t$ appears as a difference between two taps of a higher degree monomial of $\mathcal{L}_t + \mathcal{Q}_t + \mathcal{T}_t$.
- Each of the sets
$$\{34, 67, 73\},$$
$$\{2, 28, 41, 65\},$$
$$\{13, 29, 50, 64, 75\},$$
$$\{6, 14, 26, 32, 47, 61\},$$
$$\{1, 19, 27, 43, 57, 66, 78\}$$
of tap indices (all from NFSR2) of the monomials of degree $3, \ldots, 7$ of $\mathcal{L}_t + \mathcal{Q}_t + \mathcal{T}_t$ is a full positive difference set. Consequently, each two bits of the internal bitstream of NFSR2 never appear more than once as part of each (i.e., the same) of those monomials.

[44] In Grain-128a, the feedback function of the NFSR (corresponding to NFSR2 in our construction) and the output function share only a single tap (called "b_{i+95}" in [48]).

C Complexity of Finding IV Collisions

In order to show that the number of oracle queries necessary for finding an IV collision with probability $> 1/2$ exceeds 2^{60} as claimed in section 4.9, we first need to observe that, for an arbitrarily fixed key, each initial state can be the result of at most two different IVs.

Let (K, IV) and $(K, \widetilde{IV})$ with $IV \neq \widetilde{IV}$ be an IV collision, i.e., the corresponding initial states

$$\left(\left(B_0^{257}, \ldots, B_{89}^{257}\right), \left(S_0^{257}, \ldots, S_{30}^{257}\right)\right)$$

for (K, IV) and

$$\left(\left(\widetilde{B}_0^{257}, \ldots, \widetilde{B}_{89}^{257}\right), \left(\widetilde{S}_0^{257}, \ldots, \widetilde{S}_{30}^{257}\right)\right)$$

for $(K, \widetilde{IV})$ coincide. Then

$$\left(\left(B_0^{129}, \ldots, B_{89}^{129}\right), \left(S_0^{129}, \ldots, S_{30}^{129}\right)\right) = \left(\left(\widetilde{B}_0^{129}, \ldots, \widetilde{B}_{89}^{129}\right), \left(\widetilde{S}_0^{129}, \ldots, \widetilde{S}_{30}^{129}\right)\right)$$

must hold as phase 4 of the state initialization algorithm of Lizard (cf. Sec. 2.2) implements a bijective mapping over the set of all inner states. As the key is (arbitrarily) fixed, we also know that

$$\left(B_0^{128}, \ldots, B_{89}^{128}\right) = \left(\widetilde{B}_0^{128}, \ldots, \widetilde{B}_{89}^{128}\right)$$

together with either

$$\left(S_0^{128}, \ldots, S_{30}^{128}\right) = \left(\widetilde{S}_0^{128}, \ldots, \widetilde{S}_{30}^{128}\right)$$

or

$$\left(S_0^{128}, \ldots, S_{30}^{128}\right) = \left(\widetilde{S}_0^{128}, \ldots, \widetilde{S}_{30}^{128} \oplus 1\right)$$

must hold. The case

$$\left(\left(B_0^{128}, \ldots, B_{89}^{128}\right), \left(S_0^{128}, \ldots, S_{30}^{128}\right)\right) = \left(\left(\widetilde{B}_0^{128}, \ldots, \widetilde{B}_{89}^{128}\right), \left(\widetilde{S}_0^{128}, \ldots, \widetilde{S}_{30}^{128}\right)\right),$$

however, is impossible due to the fact that, like phase 4, phase 2 of the state initialization algorithm also implements a bijective mapping over the set of all inner states and, as $IV \neq \widetilde{IV}$,

$$\left(\left(B_0^0, \ldots, B_{89}^0\right), \left(S_0^0, \ldots, S_{30}^0\right)\right) \neq \left(\left(\widetilde{B}_0^0, \ldots, \widetilde{B}_{89}^0\right), \left(\widetilde{S}_0^0, \ldots, \widetilde{S}_{30}^0\right)\right).$$

This now also shows immediately that there cannot be a third key/IV combination $(K, \widehat{IV})$ with $IV \neq \widehat{IV} \neq \widetilde{IV} \neq IV$ that results in the same initial state as the IV collision (K, IV) and $(K, \widetilde{IV})$ because

$$S_{30}^{128} \neq \widetilde{S}_{30}^{128} \neq \widehat{S}_{30}^{128} \neq S_{30}^{128}$$

is a contradiction.

How many oracles queries (K, IV^i), $i = 1, \ldots$, with $IV^i \neq IV^j$ for $i \neq j$, are now necessary in order to find an IV collision with probability 1/2? Based on the above observations, we can model this as a simple urn problem. The urn contains 2^{120} different pairs of balls[45] (i.e., 2^{121} balls in total) and the question is how many balls need to be drawn (randomly and without putting them back) in order to have at least one pair with probability 1/2?

The probability p_i that at attempt $i = 1, \ldots$, a pair is completed can be upper bounded by

$$p_i \leq \frac{i-1}{2^{121} - (i-1)}.$$

Consequently, the probability $Pr[k]$ that after k attempts, at least one pair has been found, can be upper bounded by

$$Pr[k] \leq \sum_{i=1}^{k} p_i = \sum_{i=1}^{k} \frac{i-1}{2^{121} - (i-1)} < k \cdot \frac{k}{2^{121} - k}.$$

For $k \leq 2^{60}$, we have $Pr[k] < 1/2$, which proves the claim.[46]

D Implementation Details

D.1 Serialization of Phases 1 and 3 of LIZARD's State Initialization

In our implementation of Lizard, phase 1 of the state initialization is distributed over 121 clock cycles $c = 0, \ldots, 120$ as follows:

$$B_j^{0,c+1} := \begin{cases} B_{j+1}^{0,c}, & \text{for } j \in \{0, \ldots, 88\}, \\ S_0^{0,c}, & \text{for } j = 89, \end{cases}$$

$$S_i^{0,c+1} := \begin{cases} S_{i+1}^{0,c}, & \text{for } i \in \{0, \ldots, 29\}, \\ p_c, & \text{for } i = 30, \end{cases}$$

where

$$p_c := \begin{cases} K_c \oplus IV_c, & \text{for } c \in \{0, \ldots, 63\}, \\ K_c, & \text{for } c \in \{64, \ldots, 118\}, \\ K_c \oplus 1, & \text{for } c = 119, \\ 1, & \text{for } c = 120. \end{cases}$$

[45] Each pair of balls corresponds to a different pair of inner states at $t = 128$ fulfilling

$$\left(\left(B_0^{128}, \ldots, B_{89}^{128}\right), \left(S_0^{128}, \ldots, S_{30}^{128}\right)\right) = \left(\left(\widetilde{B}_0^{128}, \ldots, \widetilde{B}_{89}^{128}\right), \left(\widetilde{S}_0^{128}, \ldots, \widetilde{S}_{30}^{128} \oplus 1\right)\right).$$

[46] Using the more precise estimate $\sum_{i=1}^{k} \frac{i-1}{2^{121}-(i-1)} \leq \int_0^k \frac{x}{2^{121}-x}\,\mathrm{d}x$, one can even show that $Pr[k] \leq 1/2$ for all $k \leq 2^{60.5}$.

To avoid ambiguity, we point out that $B_j^{0,121} = B_j^0$, $j = 0, \ldots, 89$, and $S_i^{0,121} = S_i^0$, $i = 0, \ldots, 30$, where B_j^0 and S_i^0 are defined as explained in section 2.2. The initial content of the FSRs (i.e., $B_j^{0,0}$, $j = 0, \ldots, 89$, and $S_i^{0,0}$, $i = 0, \ldots, 30$) is undefined and algorithmically irrelevant as, what effectively happens here, is that the bitstring

$$K_0 \oplus IV_0, \ldots, K_{63} \oplus IV_{63}, K_{64}, \ldots, K_{118}, K_{119} \oplus 1, 1$$

is shifted into the KSGs driving registers "from the right" (in terms of Fig. 1 in Sec. 2).

Analogously, phase 3 of the state initialization is also distributed over 121 clock cycles $c = 0, \ldots, 120$ as follows:

$$B_j^{129,c+1} := \begin{cases} B_{j+1}^{129,c}, & \text{for } j \in \{0, \ldots, 88\}, \\ S_0^{129,c}, & \text{for } j = 89, \end{cases}$$

$$S_i^{129,c+1} := \begin{cases} S_{i+1}^{129,c}, & \text{for } i \in \{0, \ldots, 29\}, \\ q_c, & \text{for } i = 30, \end{cases}$$

where

$$B_j^{129,0} := B_j^{128} \text{ for } j \in \{0, \ldots, 89\},$$

$$S_i^{129,0} := S_i^{128} \text{ for } i \in \{0, \ldots, 30\},$$

$$q_c := \begin{cases} B_0^{129,c} \oplus K_c, & \text{for } c \in \{0, \ldots, 119\}, \\ 1, & \text{for } c = 120, \end{cases}$$

and B_j^{128} and S_i^{128} are defined as explained in section 2.2.

Again, to avoid ambiguity, we point out that $B_j^{129,121} = B_j^{129}$, $j = 0, \ldots, 89$, and $S_i^{129,121} = S_i^{129}$, $i = 0, \ldots, 30$, where B_j^{129} and S_i^{129} are defined as explained in section 2.2.

D.2 Module Interfaces

Listing 1.1. Verilog module port declaration for Lizard.

```
module lizard(
        input wire clk,
        input wire reset,
        input wire enable,
        input wire [0:119] key,
        input wire [0:63] iv,
        output wire keystreamBit,
        output wire keystreamFlag
        );
```

Listing 1.2. Verilog module port declaration for Grain v1.

```
module grainv1(
        input wire clk,
        input wire reset,
        input wire enable,
        input wire [0:79] key,
        input wire [0:63] iv,
        output wire keystreamBit,
        output wire keystreamFlag
        );
```

Listings 1.1 and 1.2 show the interfaces of the implemented cipher modules. For Grain v1, the straightforward implementation and the variant with serialized key/IV loading (denoted as Grain v1* in Tab. 1) have identical interfaces.

The modules use synchronous reset (taking one clock cycle) and all operations are triggered by the rising edge of the clock. By setting the `enable` flag low, the respetive cipher module can be paused at any time during state initialization or keystream generation. Setting `reset` high resets a module. Once `reset` is set low again, state initialization begins. For all modules, it is assumed that key and IV are available (via `key` and `iv`) until key/IV loading has finished. Once a cipher module has completed state initialization and enters the keystream generation phase, it changes `keystreamFlag` from low to high and outputs one keystream bit per clock cycle via `keystreamBit`.