IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2022, No. 2, pp. 1–42. DOI:10.46586/tosc.v2022.i2.1-42

The DRACO Stream Cipher

A Power-efficient Small-state Stream Cipher with Full Provable Security against TMDTO Attacks

Matthias Hamann[1], Alexander Moch[2], Matthias Krause[2] and Vasily Mikhalev[3]

[1] ERNW Research GmbH, Heidelberg, Germany
mhamann@ernw.de
[2] Universität Mannheim, Mannheim, Germany
{moch,krause}@uni-mannheim.de
[3] Universität Siegen, Siegen, Germany
vasily.mikhalev@uni-siegen.de

Abstract. Stream ciphers are vulnerable to generic time-memory-data tradeoff attacks. These attacks reduce the security level to half of the cipher's internal state size. The conventional way to handle this vulnerability is to design the cipher with an internal state twice as large as the desired security level. In lightweight cryptography and heavily resource constrained devices, a large internal state size is a big drawback for the cipher. This design principle can be found in the eSTREAM portfolio members Grain and Trivium.

Recently proposals have been made that reduce the internal state size. These ciphers distinguish between a volatile internal state and a non-volatile internal state. The volatile part would typically be updated during a state update while the non-volatile part remained constant. Cipher proposals like Sprout, Plantlet, Fruit and Atom *reuse* the secret key as non-volatile part of the cipher. However, when considering indistinguishability none of the ciphers mentioned above provides security beyond the birthday bound with regard to the volatile internal state. Partially this is due to the lack of a proper proof of security.

We present a new stream cipher proposal called Draco which implements a construction scheme called CIVK. In contrast to the ciphers mentioned above, CIVK uses the initial value and a key prefix as its non-volatile state. Draco builds upon CIVK and uses a 128-bit key and a 96-bit initial value and requires 23 % less area and 31 % less power than Grain-128a at 10 MHz. Further, we present a proof that CIVK provides full security with regard to the volatile internal state length against distinguishing attacks. This makes Draco a suitable cipher choice for ultra-lightweight devices like RFID tags.

Keywords: Symmetric-key cryptography · lightweight cryptography · stream ciphers · provable security · TMDTO attacks · Grain · RFID

1 Introduction

Stream ciphers. In symmetric key cryptography, we typically distinguish two types of encryption schemes: block ciphers and stream ciphers. Block ciphers divide a plaintext into blocks of a fixed size and encrypt one such block of data as a whole. Stream ciphers on the other hand consider the plaintext as a continuous stream of data. The stream cipher maintains an internal state and in each step it outputs one bit or several bits and updates its internal state. Throughout this work, we consider individual bit outputs. The output bit stream is then combined with the plaintext, usually using the XOR operation.

Licensed under Creative Commons License CC-BY 4.0.
Received: 2021-11-23 Revised: 2022-03-01 Accepted: 2022-05-01 Published: 2022-06-10

One advantage of stream ciphers is that their resource requirements are lower than those of block ciphers in many application scenarios. This makes them particularly useful in lightweight cryptography. Instances of stream ciphers are used in the GSM cellular phone standard (A5/1), Bluetooth (E0) and wireless networking (RC4).

Vulnerabilities. Stream ciphers are vulnerable to time-memory-data tradeoff attacks [Bab95, Gol96, BS00]. These types of attacks exploit the birthday paradox to recover an internal state. This internal state can then be used to decrypt the remaining ciphertext. Due to the birthday paradox the security of such ciphers is typically capped at half the size of the internal state. Accordingly, this has influenced the design of stream ciphers in such a way that the internal state size is at least twice the size of the desired security level. This is in stark contrast to the lightweight principle of stream ciphers, since a larger state necessarily increases resource requirements. Stream ciphers that employ a large internal state are the eSTREAM portfolio members Grain [HJM06] and Trivium [CP05]. We refer to these ciphers as the large-state-small-key construction, in short LSSK.

Recent work. Recently, efforts have been made to reduce the internal state size while still retaining a reasonable security level. Lizard [HKM17b] raises the security against key recovery attacks beyond the birthday bound, reaching a security level of $2n/3$, where n denotes the internal state's size. It does this by adding the secret key to its internal state in the last step of the state initialization. Its security against distinguishing attacks however, remains at the birthday barrier [HK15].

In addition to the volatile internal state, the stream ciphers Fruit [AGH18], Plantlet [MAM16] and Sprout [AM15] continuously use the secret key stored in non-volatile memory during their state update. The hope was that the additional key bits would enhance the security beyond the birthday bound with regard to the volatile internal state bits. However, these constructions were not equipped with a proof of security and they were eventually successfully attacked and broken [HKMZ18]. Atom [BCI+21] also uses the secret key continuously. However it does not provide beyond the birthday bound security against distinguishing attacks as the attack presented in [HKMZ18] also applies here. We refer to these ciphers as the continuous-key construction, in short CKEY.

A third proposal was recently made in [HKM17a]. Instead of continuously using the non-volatile secret key, the non-volatile initial value is employed during the state update. A proof of security was later published in [HKM19]. We refer to these ciphers as the continuous-IV construction, in short CIV.

Contribution. In this work we will present our new stream cipher proposal called Draco. Draco uses a 128-bit volatile internal state and a 128-bit non-volatile internal state. The non-volatile state consists of the initial value with a length of 96 bits and a key prefix with length 32 bits.

This new generic scheme, that uses a non-volatile state consisting of the initial value and key prefix, is called CIVK. In Section 5 we provide a security analysis in the random oracle model and we prove that CIVK provides *full* security against generic time-memory-data tradeoff attacks with regard to the volatile state length. In particular, this implies that any generic distinguishing attack against CIVK has a time complexity of $\mathcal{O}(2^{\ell_v})$, where ℓ_v denotes the volatile internal state size. In case of Draco, a time complexity of 2^{128} steps is needed for a successful distinguishing attack. The corresponding attack on CIVK can be found in Subsection 4.2 and therefore the bound shown in Section 5 is tight.

To the best of our knowledge, it is the first small-state stream cipher that achieves a full 128-bit security level against key-recovery *and* distinguishing attacks. Our main variant of Draco stores the key prefix and the IV externally. In an ultra-lightweight scenario, like RFIDs where the secret key is burned into the device or stored in an EEPROM and the frame counter is used as the IV, Draco needs 23 % less area and 31 % less power than Grain-128a at 10 MHz. The saving in power stems from reduced area requirements but particularly also from the fact that unlike previous ciphers such as Grain-128a, only half

of the state bits are constantly updated, thus significantly reducing costly dynamic power consumption.

For high-performance environments, we also present the variant DRACO$_{[\mathsf{KI}]}$ where the secret key and the initial value are stored inside the DRACO hardware module while still only 128 bits are used during the state update. At a clock speed of 1 GHz DRACO needs about 34 % less energy than Grain-128a. This demonstrates that not all internal state bits need to be constantly updated to achieve a security level of 128 bits. For more details please refer to Section 9.

DRACO is a stream cipher that operates in packet mode, i.e. there may be up to 2^{32} bits, i.e. 512 MiB, of output keystream per key-IV-pair. After this limit is reached, a new IV has to be used. No IV may be used twice. In Subsection 7.6 we argue that most transmission protocols use a packet size much lower than 512 MiB and therefore we see the packet length as a valid constraint to keystream generation.

Outline. In Section 2 we provide the basics of stream ciphers. In Section 3 we present the current stream cipher constructions with an enhanced state and introduce the CIVK construction. In Section 4 we review time-memory-data tradeoff attacks on classic stream ciphers and on CIVK. In Section 5 we present the proof of security for CIVK. In Section 6 we present the specification of the DRACO cipher. In Section 7 we argue about the choice of DRACO's parameters. In Section 8 we analyze DRACO's security against various attacks. In Section 9 we present our hardware results. Section 10 concludes this work.

2 Stream Cipher Basics

Stream ciphers are symmetric encryption algorithms intended for the online encryption of plaintext bitstreams X which have to pass an insecure channel. The encryption is performed by bitwise addition of a keystream S to X, which is generated in dependence of a secret symmetric session key k and public initial values. The legal recipient, who also knows k, decrypts the encrypted bitstream $Y = X \oplus S$ by generating S and computing $X = Y \oplus S$. In this work, we consider key stream generator-based stream ciphers, i.e. stream ciphers which generate the keystream using a so-called keystream generator (KSG).

2.1 Keystream Generation

Keystream generators are stepwise working devices that typically consist of interconnected feedback shift registers (FSRs). KSGs can be formally specified by finite automata. The KSG has an internal state length ℓ_s corresponding to the total amount of FSR register cells. The corresponding set of internal states is denoted by $Q := \{0,1\}^{\ell_s}$. In the following we describe each step of the keystream generation process.

Initially, the secret key k, the initial value x and possibly a constant C are loaded into the KSG's register cells. We refer to this state as the *loading state* $q_{\mathsf{load}}(k,x,C) \in Q$.

The state initialization algorithm computes the *initial state* $q_{\mathsf{init}}(k,x,C) \in Q$ from the loading state $q_{\mathsf{load}}(k,x,C)$.[1] This is done to ensure a sufficient level of diffusion and confusion of the initial state bits. Normally, the main component of the state initialization algorithm is an operation called *mixing*, performed by the KSG. In many stream ciphers, mixing is done by clocking the KSG multiple times without producing keystream bits.

In every clock cycle $i \geq 0$, the KSG produces an output bit $z_i = f(q_i)$ according to an *output function* $f : Q \to \{0,1\}$. Using the *state update function* $\pi : Q \to Q$, the internal state q_i is then updated to $q_{i+1} = \pi(q_i)$. Typically, π is bijective and efficiently invertible. We denote multiple evaluations of the state update function π by π^r, e.g. $q_2 = \pi^2(q_0) = \pi(\pi(q_0))$. The output key stream $S(q_0)$ is defined by concatenating all the outputs $z_0||z_1||z_2||z_3||\cdots$

[1] In the following we will omit the variables k, x and C when referring to q_{load} and q_{init}.

Some stream ciphers define an additional parameter. The *packet length* ℓ_p is defined to be the maximum count of output bits per IV x. After reaching ℓ_p output bits, the IV x is changed and state initialization is performed again.

2.2 Security Requirements

The main security requirement for stream ciphers is that it must be hard to distinguish the keystream generated on a randomly chosen secret key from a truly random bitstream. This implies the hardness of the state recovery problem:

> Let $S_{\leq \ell_p}(q)$ denote the first ℓ_p bits of keystream generated from the internal state q. For a given piece of keystream z, compute an internal state q with $z = S_{\leq \ell_p}(q)$.

This also implies that it is hard to recover the secret session key k.

The main drawback of KSG-based stream ciphers is their vulnerability to generic time-memory-data tradeoff (TMDTO) attacks [Bab95, Gol96, BS00]. These attacks allow to recover an internal state in time $2^{\ell_s/2}$ and they have a memory and data requirement of $2^{\ell_s/2}$. This reduces their effective security level to one half of the internal state size.

Moreover, several stream ciphers like Trivium have an efficiently invertible state initialization algorithm which allows to efficiently compute the secret session key from one recovered internal state. Commercial stream ciphers like $A5/1$ and Bluetooth E_0 ignored the existence of TMDTO attacks. This brought their security level below the widely accepted bound of 80 bits. The eSTREAM portfolio members Trivium [CP05] and Grain v1 [HJM06] demonstrate awareness of TMDTO attacks by being designed in accordance with the so-called LSSK construction. While for Trivium and Grain the session key length is 80 bits, the internal state lengths of Grain v1 and Trivium are 160 bits and 288 bits, respectively. This is in stark contrast to the lightweight principle of stream ciphers, since a larger state necessarily increases resource requirements.

3 Enhanced State Stream Ciphers

In the last years an intensive search for new stream cipher constructions was conducted. In particular, the goal was to decrease the internal state size while still retaining a high resistance against TMDTO attacks. This research is accompanied by the development of new information-theoretic methods that allow to prove the security of generic stream cipher constructions against TMDTO attacks. This is similar to the formal framework of Even-Mansour ciphers that was used to analyze the security of generic block cipher constructions.

Three generic stream cipher constructions have been proposed so far: (1) the Lizard construction [HKM17b], (2) the CKEY construction (underlying Fruit [AGH18], Plantlet [MAM16], Sprout [AM15] and Atom [BCI+21]), and (3) the CIV construction [HKM17a, HKM19]. The CKEY and the CIV construction rely on an enhanced state which will be explained below in Subsection 3.1. In the following, we will give an overview over these three constructions. Further we will introduce our new construction CIVK that combines the ideas of CKEY and CIV to enhance the security level.

3.1 Enhancing the Internal State

The CKEY and CIV constructions divide the internal state into a volatile part of length ℓ_v and a non-volatile part of length $\ell_{nv} := \ell_s - \ell_v$. The non-volatile memory remains unchanged during state update and state initialization. In practice, this allows to reduce the amount of costly volatile register cells.

Continuous Key. The first example was the CKEY construction. It uses the non-volatile secret key not only for initialization, as is common, but also during state initialization and keystream generation. This principle underlies the stream cipher proposals Sprout, Plantlet, Fruit and Atom. However, it was shown in [HKM19] that the resistance of the CKEY construction against generic TMDTO distinguishing attacks is at most $\ell_v/2$.
Continuous IV. The CIV construction was first proposed in [HKM17a]. Contrary to CKEY, it does not use the non-volatile key, but it uses the initial values as the non-volatile part of the internal state. This construction provides a provable security level of $\ell_v - \log_2(\ell_p)$ [HKM19]. Note that there are no stream cipher instantiations based on the CIV construction so far.
Continuous IV & Key. Our new construction CIVK uses the initial values as part of the non-volatile state as well as a prefix of length $\log_2(\ell_p)$ of the secret key. In particular, using the initial values and a key prefix from non-volatile memory, the volatile memory is initialized with the key only, with $\ell_v = \ell_k$, where ℓ_k denotes the key length. We also define the non-volatile internal state length ℓ_{nv} to be equal to the volatile state length, i.e. we have $\ell_{nv} = \ell_v = \ell_k$. The IV length ℓ_{IV} is determined by the key and packet length: $\ell_{\mathrm{IV}} = \ell_{nv} - \log_2(\ell_p) = \ell_k - \log_2(\ell_p)$.

As we will prove in this work, this allows us to reach a security level of the entire volatile internal state length ℓ_v, resp. of the entire key length ℓ_k. In the next subsection we will specify the CIVK construction in detail. The proof can be found in Section 5, the resulting bound can be found in Subsubsection 5.2.4.

3.2 The CIVK Construction

We denote by $\mathcal{Q}_{nv}$ the non-volatile internal state space and by $\mathcal{Q}_v$ the volatile internal state space. We denote an internal state by $\langle a \,|\, b \rangle$, where the left side $a \in \mathcal{Q}_{nv}$ denotes the non-volatile internal state and the right side $b \in \mathcal{Q}_v$ denotes the volatile internal state. We chose this notation to make it easier to distinguish internal states from arbitrary tuples. Also, if for example the non-volatile state a consists of two parts a_1 and a_2, we will denote this state by $\langle a_1, a_2 \,|\, b \rangle$. The key space is denoted by $\mathcal{K}$ and the IV space is denoted by $\mathcal{IV}$. The following description will define the keystream generation using the CIVK construction.

Packet length. CIVK is a construction that works in packet mode. The parameter ℓ_p defines the packet length. For each key-IV-pair $(k, x) \in \mathcal{K} \times \mathcal{IV}$ CIVK may output up to ℓ_p keystream bits.

Enhanced State. The internal state is divided into a volatile part that is updated during state updates and a non-volatile part that is *not* updated during state updates. The volatile part of the loading state consists of the secret key k only. The non-volatile part consists of the IV x and a key prefix k^{pre} of length at least $\log(\ell_p)$. For a given key-IV-pair $(k, x) \in \mathcal{K} \times \mathcal{IV}$ the loading state of CIVK is denoted by $\langle x, k^{\mathsf{pre}} \,|\, k \rangle$. Typically the IV x and the key prefix k^{pre} will be concatenated.

State Size. The parameters ℓ_k and ℓ_p are to be specified by the respective cipher built upon CIVK. The length of the non-volatile internal state is equal to the key length, i.e. $\ell_v = \ell_k$. The key prefix has a length of at least $\log(\ell_p)$ as this allows to improve upon the CIV construction to reach a security level of 2^{ℓ_k} instead of $2^{\ell_k}/\ell_p$. Further, we want to be able to generate 2^{ℓ_k} bits of keystream per key k. As there is a limit of ℓ_p bits per key-IV-pair (x, k), the IV length has to be at least $\log(2^{\ell_k} - \ell_p)$. This will ensure that the non-volatile state length ℓ_{nv} has at least the size of the volatile state, i.e. $\ell_{nv} \geq \ell_v = \ell_k$.

Mixing Function. The loading state $\langle x, k^{\mathsf{pre}} \,|\, k \rangle$ is used as input to the mixing function p. Its task is to provide the initial state with enough confusion and diffusion for further

operation and corresponds to clocking the cipher without producing output bits.

$$p : \mathcal{Q}_{nv} \times \mathcal{Q}_v \to \mathcal{Q}_{nv} \times \mathcal{Q}_v, \langle x, k^{\mathsf{pre}} \,|\, k \rangle \mapsto \langle x, k^{\mathsf{pre}} \,|\, y \rangle.$$

State Update. The state update function π updates an internal state $\langle x, k^{\mathsf{pre}} \,|\, y \rangle$ to the next internal state $\langle x, k^{\mathsf{pre}} \,|\, y' \rangle$.

$$\pi : \mathcal{Q}_{nv} \times \mathcal{Q}_v \to \mathcal{Q}_{nv} \times \mathcal{Q}_v, \langle x, k^{\mathsf{pre}} \,|\, y \rangle \mapsto \langle x, k^{\mathsf{pre}} \,|\, y' \rangle.$$

r successive invocations of the state update function π on an internal state $\langle x, k^{\mathsf{pre}} \,|\, y \rangle$ are denoted by $\pi^r \langle x, k^{\mathsf{pre}} \,|\, y \rangle$, e.g. for three successive invocations we write:

$$\pi^3 \langle x, k^{\mathsf{pre}} \,|\, y \rangle = \pi(\pi(\pi \langle x, k^{\mathsf{pre}} \,|\, y \rangle)).$$

It is needed that the period of the state update function π is larger than ℓ_p for the entire internal state space. This means that for any internal state $\langle x, k^{\mathsf{pre}} \,|\, y \rangle$ the set $\{\langle x, k^{\mathsf{pre}} \,|\, y \rangle, \pi^1 \langle x, k^{\mathsf{pre}} \,|\, y \rangle, \ldots, \pi^{\ell_p - 1} \langle x, k^{\mathsf{pre}} \,|\, y \rangle\}$ contains ℓ_p distinct elements.

Output Function. The output function f maps an internal state $\langle x, k^{\mathsf{pre}} \,|\, y \rangle$ to an output bit $z \in \{0, 1\}$.

$$f : \mathcal{Q}_{nv} \times \mathcal{Q}_v \to \{0, 1\}, \langle x, k^{\mathsf{pre}} \,|\, y \rangle \mapsto z.$$

Keystream Generation. Let (k, x) be an arbitrary key-IV-pair and let k^{pre} be the corresponding key prefix. Using the functions defined above we can define the construction function e that corresponds to the keystream generation using the CIVK construction:

$$e : \mathcal{K} \times \mathcal{IV} \times \{0, \ldots, \ell_p - 1\} \to \{0, 1\}, (k, x, r) \mapsto f(\pi^r(p\langle x, k^{\mathsf{pre}} \,|\, k \rangle)).$$

We consider individual output bits and e outputs the r-th keystream bit. The entire keystream packet of length ℓ_p for a key-IV-pair (k, x) looks as follows:

$$e(k, x, 0) \,||\, e(k, x, 1) \,||\, \cdots \,||\, e(k, x, \ell_p - 2) \,||\, e(k, x, \ell_p - 1).$$

3.2.1 Comments on the State Length

The volatile loading state consists of the key k only and therefore we have $\ell_v = \ell_k$, where ℓ_k is regarded as fixed. The length of the key prefix k^{pre} and the IV x are not fixed but rather lower bounded as described above. In the following we will elaborate on the consequences of smaller or larger lengths.

If the length of the non-volatile state were lower than that of the volatile state, the attack by Babbage and Golić presented in Subsection 4.2 would yield a smaller complexity than 2^{ℓ_k}.

The key prefix allows to improve upon the bound of CIV and is hence chosen to be at least $\log(\ell_p)$. A smaller value will decrease security as it makes the second attack in Subsection 4.2 more probable.[2] A longer key prefix without compromising the IV length will increase the complexities of both attacks in in Subsection 4.2. Yet, the exhaustive key search remains with a complexity of 2^{ℓ_k}.

The IV length mainly influences how many keystream bits per key can be generated. With a longer IV the second attack presented in Subsection 4.2 would still yield a complexity of 2^{ℓ_k}. Theoretically this would allow for more than 2^{ℓ_k} bits of keystream to be generated. A smaller IV would reduce the total amount of keystream bits that can be generated per key. Depending on the size of the reduction this may not be an issue.

[2] The first attack also has a lower complexity if the non-volatile state length is decreased by a decrease in the key prefix length.

3.3 Hardware Implications of Continuous IV Access

Now one may argue from a hardware perspective that while the secret key has to be stored anyhow (e.g., also for LSSK stream ciphers such as Trivium, Grain etc.) in order to be reused with other IVs, this would not be the case for the IV. Hence, at first sight, assuming that the IV is still accessible after state initialization might be considered cheating. However, we do not think that this is the case for many application scenarios. For example, in A5/1 of the GSM standard, the IV used for encrypting a data packet is the respective (sequentially incremented) 22-bit frame number. Hence, any A5/1 device needs some memory containing this frame number anyhow. Similarly, the DECT standard for cordless telephone systems relies on frame numbers as IVs for encryption.

In general, especially for ciphers with small IV spaces, there always has to be a mechanism like a stepwise incremented IV storage to make sure that the same IV is not accidentally used twice under the same secret key. Similarly, in all communication scenarios like A5/1 or DECT, where the packet number serves at the same time as an IV source, one will always have this information. Note that such packet counters are not only prevalent for standard network transmission protocols such as TCP/IP, but are also a common component of lightweight wireless devices such as RFID tags, e.g., for synchronization purposes and in order to protect against replay attacks.

The vast majority of RFID tags are based on ASICs (application-specific integrated circuits), whose primary types of writable storage are non-volatile EEPROMs and volatile flip-flops. In [MAM16], the designers of Plantlet extensively studied the effects of continuously reading the secret key from an EEPROM during keystream generation. Despite certain drawbacks of this approach, such as an increased design complexity and a potential reduction of the maximal achievable throughput, they concluded that this is in fact feasible. This naturally holds for any other kind of data stored in an EEPROM, too, such as a packet counter serving as the IV source and being accessed in the same way. In particular, the key-IV-schedule of Draco (cf. Section 6) accesses the 96 IV bits and the bits of the 32-bit key prefix in sequential order, just like the key schedule of Plantlet accesses the cipher's 80-bit key in sequential order.

The designers of Plantlet found this to be beneficial in terms of limiting the negative performance impact of continuous EEPROM access on the maximal achievable throughput. If the storage location of the IV source (such as the aforementioned network packet counter) is an array of flip-flops instead, the feasibility of continuous IV access is straightforward, because the ASIC's cryptographic logic can be connected to those flip-flops through wires at practically no cost.

In the previous paragraphs, we have explained that there are various scenarios where the cipher's key and/or IV are actually already present in some storage location on the device, allowing to *reuse* this when realizing continuous key-IV access. In fact, the resulting savings on flip-flops (and, thus, chip area) inside the cipher module have long been the major motivation for such designs. In Section 9, we show that due to this focus on the *number* of flip-flops, ignoring their actual *usage*, a great potential for reducing power consumption has been missed, so far.

More precisely, with our implementation variant $\text{Draco}_{[\text{KI}]}$ we demonstrate that even if a $2n$-bit storage is required inside the cipher hardware module to achieve n-bit security against TMDTO attacks, algorithmically keeping half of this state constant is much more efficient (cf. Tab. 2 in Section 9) than and equally secure (see Section 8 and Section 5) as constantly updating the whole of it. That is, we show that even if the key and the IV are stored locally inside of the cipher hardware module (thus eliminating all the scenario assumptions / usage restrictions described above) in order to implement continuous key-IV access, our new small-state stream cipher Draco still allows to save up to 34 % of energy as compared to Grain-128a when producing 10 kbit of keystream (including state initialization) at a clock speed of 1 GHz.

4 Time-Memory-Data Tradeoff Attacks

TMDTO attacks are generic attacks that only have black-box access to the mixing algorithm and the output function of the KSG. The attacks assume that these components are ideally designed. This means in particular that no internals of the underlying components can be exploited by the adversary. The goal of TMDTO attacks is to discover weaknesses in how these components interact to produce the keystream.

TMDTO attacks are often divided into a precomputation phase and an online phase. In the precomputation phase some helping data structure is computed. In the online phase the attack is executed based on the available keystream and the helping data structure. Four cost dimensions are relevant to these attacks:

- The amount of keystream available D in the online phase.
- The time consumption T of the online phase.
- The time consumption P of the precomputation phase.
- The total memory consumption M of precomputation and online phase.

The costs are expressed in a so-called tradeoff curve. It consists of all 4-tuples (T, M, D, P) of cost values that allow for a successful attack with high probability. For attacks without a precomputation phase, the cost dimension P is not considered.

The first TMDTO attacks against KSG-based stream ciphers go back to Babbage [Bab95] and Golić [Gol96]. They yield the tradeoff curve $T \cdot D = 2^{\ell_s}$. In particular, for $T = D = 2^{\ell_s/2}$, this caps the security of stream ciphers at the birthday bound. We describe the idea of these attacks in Subsection 4.1.

Biryukov and Shamir [BS00] combined the attacks of Babbage and Golić with Hellman's attack on block ciphers [Hel80]. This yields the tradeoff curve $T \cdot M^2 \cdot D^2 = 2^{2 \cdot \ell_s}$ with $P = 2^{\ell_s}/D$. In [BS00], Biryukov and Shamir also discuss a technique called BSW-sampling, which was originally used by Biryukov, Shamir, and Wagner in [BSW01] to attack the GSM cipher A5/1. While BSW-sampling allows to relax the restriction $T \geq D^2$ in the above attack, the tradeoff curve $T \cdot M^2 \cdot D^2 = 2^{2 \cdot \ell_s}$ and the relation $P = 2^{\ell_s}/D$ remain unchanged. Hence, if one considers precomputation to be part of the overall attack complexity (as we do[3]), even the use of BSW-sampling does not allow for attacks with overall complexity lower than $2^{\ell_s/2}$. Moreover, the applicability of BSW-sampling is highly cipher specific (see [BS00] for further details) and, thus, corresponding TMDTO attacks are not fully generic.

In [HS05], Hong and Sarkar consider the TMDTO case of sampling pairs of keys and IVs instead of sampling from the space of internal states. In a single-key scenario, as treated in our analysis, this approach has an overall complexity (including precomputation) at least as large the complexity of exhaustive key search. Only in scenarios with multiple keys, where the attacker's goal is to discover *one* of these keys, a lower overall complexity can be achieved. Consequently, neither of the results in [BS00] and [HS05] conflicts with our security bounds for CKEY and CIVK.

In the remainder of this section we first sketch the original attacks by Babbage and Golić. Then we show how to modify and apply these attacks the CIVK construction, in order to derive a corresponding upper bound for security against TMDTO attacks. Together with the results presented in Subsubsection 5.2.4 this will yield a tight bound for CIVK.

[3] In [Hel80], Hellman himself acknowledges that "[i]n the real world, we must ensure that the precomputation is not excessive". Our new stream cipher Draco introduced in Section 6 targets a security level of 128 bits and a precomputation complexity of 2^{128} is clearly 'excessive'. After all, e.g., AES could otherwise be attacked with online complexity around 2^{85} using Hellman's original time-memory tradeoff.

4.1 The TMDTO Attack of Babbage and Golić

Suppose that the attacker knows a set $\mathcal{S}$ of D keystream blocks of length ℓ_s. These blocks originate from one session with the secret session key k, but are allowed to stem from different keystream packets. Let $R = \{q^1, \ldots, q^D\}$ denote the set of corresponding internal states. The attacker generates a set of T internal state/keystream block pairs $(y, S_{\leq \ell_s}(y))$ for randomly chosen internal states $y \in Q$. If $D \cdot T \approx 2^{\ell_s}$, there will be a collision between the observed keystream and the generated keystream with high probability according to the birthday paradox. In particular for $D = T = 2^{\ell_s/2}$ the security is capped at the birthday bound. As a result, the attacker knows the internal state q^j corresponding to one keystream block of a packet generated with respect to a known initial value x. This allows the adversary to compute the entire keystream packet corresponding to k and x as well as the initial state q_{init} for this packet. Knowing an internal state, the adversary can simply compute the following internal states using the known state update function and use these states as inputs to the output function to generate the remaining keystream. Moreover, for Trivium, Grain v1, and many other ciphers, it is possible to efficiently recover the secret key k from q_{init}.

4.2 TMDTO Attacks against CIVK

The CIVK construction refers to a cipher working in packet mode. The packet length is ℓ_p. In CIVK the IV x and a prefix of length $\log_2(\ell_p)$ of the secret session key k are continuously employed during mixing and keystream generation. Thus the IV and the key prefix become the non-volatile part of the cipher's internal state. The secret key is loaded into the volatile part of the internal state.

Note that there are two ways of applying the Babbage-Golić TMDTO attack to this cipher. The first approach is to mount the attack in its original form, which does not take the special structure of the internal states into account. This attack has the tradeoff curve $T \cdot D = 2^{\ell_s} = 2^{\ell_{nv}+\ell_v}$. Let ℓ_{IV} be the length of the IV x. The maximum amount of data D that can be obtained is

$$\ell_p \cdot 2^{\ell_{\mathrm{IV}}} = 2^{\log_2(\ell_p)+\ell_{\mathrm{IV}}} = 2^{\ell_{nv}}$$

as at most ℓ_p bits per IV x are produced. For this maximal D the tradeoff curve yields a time complexity of $T = 2^{\ell_v} = 2^{\ell_k}$.

The second approach is to make use of the fact that the IVs for the keystream packets are known by the adversary. We assume that the adversary obtains p keystream packets (of length ℓ_p) corresponding to the initial values x_1 to x_p. This corresponds to a data complexity of $D = p \cdot \ell_p$. For each x_i the attacker generates s times a random key prefix k_i^{pre} *and* a random volatile internal state $z_i \in \{0,1\}^{\ell_v}$. From x_i, k_i^{pre} and z_i the attacker computes an output keystream block $S_{\leq \ell_s}(x, \tilde{k}, z)$ of length ℓ_s.

A collision in the *volatile* internal state occurs with high probability if $D \cdot s = 2^{\ell_v}$. Additionally, the adversary needs a correct *non-volatile* internal state. The IV is known to the attacker and the key prefix is guessed correctly with a probability of ℓ_p^{-1}. Hence, the probability that one out of the s generated key prefixes per IV x_i is correct, is s/ℓ_p. We obtain that an attack is successful if

$$D \cdot \frac{s}{\ell_p} = p \cdot \ell_p \cdot \frac{s}{\ell_p} = p \cdot s = 2^{\ell_v}.$$

Note that regardless of how many keystream packets are observed, the attacker always has a time complexity of $T = p \cdot s = 2^{\ell_v}$. Insofar, it would be ideal to only attack one keystream packet.

Also note that, assuming a chosen-IV attacker, all $p \cdot s = 2^{\ell_k}$ keystream packets can be precomputed and need to be stored in efficiently searchable data structure, i.e. a binary

tree. This will require a time complexity of 2^{ℓ_k} in the offline phase and a space (memory) complexity of 2^{ℓ_k}. The online phase, when observing a keystream packet, will then have a time complexity of $\log_2(2^{\ell_k}) = \ell_k$ to search for the collision on the previously computed keystream packets. While the online time complexity is significantly reduced, there is an excessive amount of precomputation time necessary and the space complexity in the online phase is 2^{ℓ_k}. Even if a time complexity of 2^{ℓ_k} were not excessive, a space complexity of 2^{ℓ_k} may very well be.

5 Proof of Security

In the following, we will present the preliminaries, notation and random oracle model necessary for the proof of security.

5.1 Random Oracle Model

An adversary will be interacting with a set of three oracles in one of two worlds: the real world or the ideal world. There will be an oracle P for the mixing function, an oracle F for the output function and an oracle E for the construction function.[4] The adversary can query the oracles with the inputs of the respective functions and it will receive answers from the oracle. These query-answer-pairs are collected by the adversary in a transcript.

The P- and F-oracles will answer their queries using ideal randomized primitives in either world: The P-oracle will use a random permutation $\mathbf{P}$ (more specifically: multiple random permutations as described in Subsubsection 5.1.3) and the F-oracle will use a random function $\mathbf{F}$. In the real world, the E-oracle uses $\mathbf{P}$ and $\mathbf{F}$ as underlying building blocks. In the ideal world, the E-oracle will have access to another independent random function $\mathbf{E}$. By assuming the underlying building blocks to be ideal one can abstract from possible weaknesses in the mixing function and the output function that an implementation may have and show that the scheme, the interaction of those building blocks, is secure. This will *not* prove an instantiation to be secure but provide a plausible justification for the structure of a cipher built upon CIVK, in this case Draco.

In the ideal world $\mathbf{E}$, corresponding to the encryption function, will sample the output bits uniformly at random from $\{0, 1\}$. We will show that the adversary can not distinguish the ideal world from the real world in this scenario. In particular, this will show that the keystream generated by the "real" encryption function is indistinguishable from a truly random bitstream.

We will first describe the proof technique, then the distinguishing game and then explain in detail how the oracle queries and the adversary's transcript look like.

5.1.1 H-coefficient Technique

To prove the security of CIVK we will use the H-coefficients technique. The H-coefficients technique [Pat08] is a proof method due to Patarin, where we consider the variant by Chen and Steinberger [CS14]. The results of the interaction of an adversary $\mathbf{A}$ with its oracles are collected in a transcript τ. The oracles can sample randomness prior to the interaction (often a key or an ideal primitive that is sampled beforehand), and are then deterministic throughout the experiment [CS14]. The task of $\mathbf{A}$ is to distinguish the real world $\mathcal{O}_{\text{real}}$ from the ideal world $\mathcal{O}_{\text{ideal}}$. Let Θ_{real} and Θ_{ideal} denote the distribution of transcripts in the real and the ideal world, respectively. A transcript τ is called *attainable* if the probability to obtain τ in the ideal world – i.e. over Θ_{ideal} – is non-zero. Then, the fundamental Lemma of the H-coefficients technique, the proof to which is given in [CS14, Pat08], states:

[4] π is publicly known and hence no oracle is needed.

Table 1: Inputs and outputs of the four oracle query types.

Query Type	Input	Output
P	$\langle x_P, k_P^{\mathsf{pre}} \mid k_P \rangle$	$\langle x_P, k_P^{\mathsf{pre}} \mid y_P \rangle$
P^{-1}	$\langle x_P, k_P^{\mathsf{pre}} \mid y_P \rangle$	$\langle x_P, k_P^{\mathsf{pre}} \mid k_P \rangle$
F	$\langle x_F, k_F^{\mathsf{pre}} \mid y_F \rangle$	$z_F \in \{0,1\}$
E	(x_E, r_E)	$z_E \in \{0,1\}$

Lemma 1 (Fundamental Lemma of the H-coefficient Technique [Pat08])**.** Assume, the set of attainable transcripts can be partitioned into two disjoint sets GoodT and BadT. Further assume that there exist $\epsilon_1, \epsilon_2 \geq 0$ such that for any transcript $\tau \in \mathsf{GoodT}$, it holds that

$$\frac{\Pr[\Theta_{\text{real}} = \tau]}{\Pr[\Theta_{\text{ideal}} = \tau]} \geq 1 - \epsilon_1, \quad \text{and} \quad \Pr[\Theta_{\text{ideal}} \in \mathsf{BadT}] \leq \epsilon_2.$$

Then, for all adversaries $\mathbf{A}$, it holds that $\mathbf{A}$'s distinguishing advantage $\Delta_{\mathbf{A}}(\mathcal{O}_{\text{real}}, \mathcal{O}_{\text{ideal}})$ can be upper bounded by $\Delta_{\mathbf{A}}(\mathcal{O}_{\text{real}}, \mathcal{O}_{\text{ideal}}) \leq \epsilon_1 + \epsilon_2$.

5.1.2 The Distinguishing Game

In the beginning of the adversary's interaction with the oracles a key $k \xleftarrow{\$} \mathcal{K}$ will be sampled uniformly at random from the key space $\mathcal{K}$. Next, the adversary poses its questions to the P-, F- and E-oracles with the additional limit of at most ℓ_p E-queries per IV x. All of the query-answer-pairs will be collected in the corresponding τ_P, τ_F and τ_E transcripts. When the adversary is finished with its interaction with the oracles it is given the secret key k as well as the transcript τ_α, which contains the intermediate values generated during an E-query and will be defined more explicitly in Subsubsection 5.1.4. Based on the transcript $\tau = (\tau_P, \tau_E, \tau_F, \tau_\alpha, k)$ the adversary has to make its decision whether it was interacting with the real world or the ideal world and output a decision bit. If the adversary's guess is correct, it wins the game. Our task is to upper bound the adversary's success probability.

5.1.3 Oracle Queries

The adversary will be given access to the P-, F- and E-oracles that correspond to the mixing function, output function and construction function respectively. For clarity, we provide a table about how the inputs, as chosen by the adversary, and outputs of the respective queries look like and then we will explain how the oracles are implemented. Note that we index the variables with the query type. In particular k_P and k_F denote *key guesses* and need not be equal to the actual key k. We chose this notation to make it clear that when using k_P, we refer to the variable that is used in place of the actual key k in a P-query.

P-oracle. The P-oracle that corresponds to the mixing function will be implemented using random permutations. Note that the mixing function p keeps the non-volatile internal state unchanged and only the volatile internal state gets permuted. Hence, for every $(x, k^{\mathsf{pre}}) \in \mathcal{Q}_{nv}$ the volatile internal state is permuted using an independent random permutation $\mathbf{P}_{x,k^{\mathsf{pre}}} : \mathcal{Q}_v \to \mathcal{Q}_v$. For each $\mathbf{P}_{x,k^{\mathsf{pre}}}$ we will use lazy sampling: As the first P-query $\langle x_P, k_P^{\mathsf{pre}} \mid k_P \rangle$ arrives, the oracle will sample the answer uniformly at random from all 2^{ℓ_v} possible answers. As the second P-query arrives, the oracle will sample the answer uniformly at random from all $2^{\ell_v} - 1$ remaining possible answers, and so on. The permutation $\mathbf{P}$ is defined as follows:

$$\mathbf{P} : \mathcal{Q}_{nv} \times \mathcal{Q}_v \to \mathcal{Q}_{nv} \times \mathcal{Q}_v, \langle x_P, k_P^{\mathsf{pre}} \mid k_P \rangle \mapsto \langle x_P, k_P^{\mathsf{pre}} \mid \mathbf{P}_{x_P, k_P^{\mathsf{pre}}}(k_P) \rangle$$

The adversary may query the P-oracle in either the forward or the backward direction.

Remark 1. In either world it is possible to choose k_P^{pre} and k_P such that k_P^{pre} is *not* a prefix of k_P. As the mixing function is bijective, the corresponding internal state $\langle x_P, k_P^{\mathsf{pre}} \,|\, y_P\rangle$ will be invalid, i.e. it will not occur during an actual encryption using CIVK. These states cannot be used to obtain a distinguishing event. We will ignore queries of this type as they yield no advantage to the adversary.

F-oracle. The F-oracle that corresponds to the output function will be implemented using a random function $\mathbf{F} : \mathcal{Q}_{nv} \times \mathcal{Q}_v \to \{0,1\}$. Only queries in the forward direction are allowed. We again use lazy sampling; as an F-query arrives the output is sampled uniformly at random from $\{0,1\}$.

E-oracle. The E-oracle is defined differently in the real world and in the ideal world: In the real world it is defined similarly to the construction function e and implicitly uses the secret key k, the random permutation $\mathbf{P}$, the state update function π and the random function $\mathbf{F}$:

$$\mathbf{E} : \mathcal{IV} \times \{0, \ldots, \ell_p - 1\} \to \{0,1\}, (x_E, r_E) \mapsto \mathbf{F}(\pi^{r_E}(\mathbf{P}\langle x_E, k^{\mathsf{pre}} \,|\, k\rangle))$$

In the ideal world $\mathbf{E} : \mathcal{IV} \times \{0, \ldots, \ell_p - 1\} \to \{0,1\}$ is a random function with outputs sampled uniformly at random from $\{0,1\}$.

5.1.4 Transcripts

All of the adversary's queries to the oracles and the corresponding answers will be collected in a transcript τ. In particular, we will keep separate transcripts τ_P, τ_F and τ_E for each of the corresponding oracles P, F and E defined as follows:

$$\tau_P := \{(x_P, k_P^{\mathsf{pre}}, k_P, y_P) \mid \langle x_P, k_P^{\mathsf{pre}} \,|\, k_P\rangle \text{ is a } P\text{-query and } \langle x_P, k_P^{\mathsf{pre}} \,|\, y_P\rangle \text{ its answer.}\}$$
$$\tau_F := \{(x_F, k_F^{\mathsf{pre}}, k_F, z_F) \mid \langle x_F, k_F^{\mathsf{pre}} \,|\, k_F\rangle \text{ is an } F\text{-query and } z_F \text{ its answer.}\}$$
$$\tau_E := \{(x_E, r_E, z_E) \mid (x_E, r_E) \text{ is an } E\text{-query and } z_E \text{ its answer.}\}$$

The transcripts mentioned above are visible to the adversary as it makes its queries to the oracles. Once the adversary's interaction with the oracles is finished it will additionally be given the secret key k and the transcript τ_α defined as follows:

$$\tau_\alpha := \{(x_E, r_E, \alpha_E, \alpha_E^r) \mid (x_E, r_E) \text{ is an } E\text{-query and } (\alpha_E, \alpha_E^r) \text{ its } \alpha\text{-values.}\}$$

The α-values for each E-query (x_E, r_E) are defined as follows:

$$\alpha_E := P\langle x_E, k^{\mathsf{pre}} \,|\, k\rangle \quad \text{and} \quad \alpha_E^{r_E} := \pi^{r_E}(P\langle x_E, k^{\mathsf{pre}} \,|\, k\rangle)$$

The α-values correspond to the internal states that are generated during an E-query. α_E represents the initial value and $\alpha_E^{r_E}$ represents the input to the output function $\mathbf{F}$. We will also sample these values in the ideal world, even though the construction function E does not depend on these.

The full transcript can be written as a 5-tuple $\tau = (\tau_P, \tau_E, \tau_F, \tau_\alpha, k)$.

5.2 Security Analysis

We will now perform the security analysis of CIVK. In particular, for the CIVK construction we will show the following:

Lemma 2. *For the* CIVK *construction as defined in Subsection 3.2 where the mixing function p and the output function f are replaced with their ideal counterparts, as defined in 5.1.3, and up to ℓ_p adversarial queries per IV $x \in \mathcal{IV}$ we have that for all adversaries* $\mathbf{A}$ *it holds that*

$$\underset{\mathbf{A}}{\Delta}(\mathcal{O}_{real}, \mathcal{O}_{ideal}) \leq \frac{q}{2^{\ell_k} - q}.$$

From an attacker's point of view, the goal is to recover an internal state by obtaining a collision between an observed keystream and a generated keystream. If the keystreams collide, there is a high probability that they were generated from the same internal state. In particular, the same internal state always generates the same keystream; in the real world.

In our model, the adversary does *not* have to rely on observing collisions in the keystream as it is provided the α-values at the end of the interaction. The α-values correspond to the internal states of the respective output bit. Hence, the adversary's goal will be to obtain a collision between the α-values and the inputs of the F-queries.

If there occurs a collision between an α-value (corresponding to an E-query) and an F-query the corresponding outputs of the E-query and the F-query will always be identical *in the real world*, as $\mathbf{E}$ internally uses $\mathbf{P}$, π and $\mathbf{F}$. *In the ideal world* however, $\mathbf{E}$ is another random function independent of $\mathbf{P}$, π and $\mathbf{F}$ and hence colliding α-values and F-query inputs only lead to the same output bit with a probability of $\frac{1}{2}$. This will be used as a distinguishing, i.e. bad, event.

5.2.1 Bad Events

Note that the IV x is chosen by the adversary. There are two strategies for the adversary to obtain a collision between the α-values and an F-query input:

1. Guess the key prefix k^{pre} as well as a volatile internal state y correctly and ask the corresponding F-query $F\langle x, k^{\mathsf{pre}} \,|\, y\rangle$.
2. Guess the key k correctly, ask the corresponding P-query $P\langle x, k^{\mathsf{pre}} \,|\, k\rangle$ to obtain $\langle x, k^{\mathsf{pre}} \,|\, y\rangle$ and ask the corresponding F-query $F\langle x, k^{\mathsf{pre}} \,|\, y\rangle$.

If the outputs of the F-query and the output of the E-query corresponding to the colliding α-value differ, the adversary surely is in the ideal world. We will introduce two bad events that correspond to the two strategies mentioned above.

bad_1 (Guessing the key prefix and an internal state) There exists an E-query (x_E, r_E) and an F-query $\langle x_F, k_F^{\mathsf{pre}} \,|\, y_F\rangle$ such that

$$\alpha_E^{r_E} = \langle x_F, k^{\mathsf{pre}} \,|\, y_F\rangle \text{ and } E(x_E, r_E) \neq F\langle x_F, k^{\mathsf{pre}} \,|\, y_F\rangle.$$

bad_2 (Guessing the key) There exists a P-query $\langle x_P, k_P^{\mathsf{pre}} \,|\, k_P\rangle$, an E-query (x_P, r_E) and an F-query $\pi^{r_E}(P\langle x_P, k_P^{\mathsf{pre}} \,|\, k_P\rangle)$ such that

$$(k_P^{\mathsf{pre}}, k_P) = (k^{\mathsf{pre}}, k) \text{ and } E(x_P, r_E) \neq F(\pi^{r_E}(P\langle x_P, k_P^{\mathsf{pre}} \,|\, k_P\rangle)).$$

Remark 2. Note that bad_2 represents a special case of bad_1. In particular we have that

$$P\langle x_P, k_P^{\mathsf{pre}} \,|\, k_P\rangle = \alpha_E \text{ and } \pi^{r_E}(P\langle x_P, k_P^{\mathsf{pre}} \,|\, k_P\rangle) = \alpha_E^{r_E}.$$

Considering these as separate bad events will simplify our analysis.

5.2.2 Bounding the Bad Events

Lemma 3.

$$\Pr\left[\Theta_{ideal} \in \mathit{BadT}\right] \leq \frac{q}{2^{\ell_k} - q}.$$

Proof. Since we are in the ideal world, the answers to the E-, F- and P-queries are independent of the secret key (k^{pre}, k). For simplicity, we will sample the secret key (k^{pre}, k) after the adversary's interaction with the oracles. Also, we will sample the values α_E for each query after the adversary's interaction with the oracles.

bad_2: We will first consider the event bad_2. There are at most q P-queries with at most q distinct $(k_P^{\mathsf{pre}}, k_P)$. The secret key (k^{pre}, k) is sampled independently at random with regard to the uniform distribution from the set $\{0,1\}^{\ell_k}$. The probability of a collision with the secret key (k^{pre}, k) can therefore be upper bounded by $q/2^{\ell_k}$.

The outputs of the E- and F-queries collide with a probability of $1/2$. We obtain:

$$\Pr[\mathsf{bad}_2] \leq \frac{1}{2} \cdot \frac{q}{2^{\ell_k}} \leq \frac{1}{2} \cdot \frac{q}{2^{\ell_k} - q}.$$

bad_1: We will now consider the event bad_1. Since we already bounded the probability for bad_2, we will now consider $\Pr[\mathsf{bad}_1 | \neg\mathsf{bad}_2]$. We decided to sample α_E after the adversary's interaction with the oracles.

By conditioning on $\neg\mathsf{bad}_2$, we know that for all $x \in \mathcal{IV}$ no value $P\langle x, k^{\mathsf{pre}} \,|\, k\rangle$ has yet been sampled. This applies since there was no adversarial P-query with the correct key. This implies, as we sample the α-values after the adversary's interaction with the oracle, no α-value has yet been sampled.

To obtain a collision between an F-query and an α-value, three conditions need to apply:

1. The key prefix k^{pre} needs to be guessed correctly.
2. The F-query and the E-query, resp. α-value, need to utilize the same IV x.
3. A collision in the volatile state needs to occur.

We will individually bound the conditions above. Fix any F-query $F\langle x_F, k_F^{\mathsf{pre}} \,|\, y_F\rangle$.

1. As we sample the key k after the adversary's interaction with the oracles, a key prefix collision occurs with a probability of
$$\Pr[k_F^{\mathsf{pre}} = k^{\mathsf{pre}}] = \ell_p^{-1}.$$
2. There are at most ℓ_p E-queries, resp. α-values, to collide with, as the E-queries are limited to ℓ_p queries per IV $x \in \mathcal{IV}$.
3. Fix some E-query (x_E, r_E) where $x_E = x_F$. Consider the internal state $\rho = \pi^{-r_E}\langle x_F, k^{\mathsf{pre}} | y_F\rangle$. The value $\alpha_E = P\langle x_E, k^{\mathsf{pre}} \,|\, k\rangle$ is sampled after the F-query. As there are at most q queries we obtain
$$\Pr[P\langle x_E, k^{\mathsf{pre}} \,|\, k\rangle = \rho] \leq (2^{\ell_v} - q)^{-1}.$$

Note that there could be P-queries utilizing x_E and k^{pre} with $k_P \neq k$ and therefore we need to upper bound the amount of queries above by q. We obtain that the probability of a single fixed F-query to collide with an α-value is upper bounded by:

$$\Pr[k_F^{\mathsf{pre}} = k^{\mathsf{pre}}] \cdot \ell_p \cdot \Pr[P\langle x_E, k^{\mathsf{pre}} \,|\, k\rangle = \rho] = \frac{1}{\ell_p} \cdot \frac{\ell_p}{2^{\ell_v} - q} = \frac{1}{2^{\ell_v} - q}.$$

The outputs of the E- and F-queries collide with a probability of $1/2$. Summing up over at most q F-queries we obtain:

$$\Pr[\mathsf{bad}_1 \mid \neg\mathsf{bad}_2] \leq \frac{1}{2} \cdot \frac{q}{2^{\ell_v} - q}.$$

Lemma 3 follows from the union bound of the above individual bad events. Note that $\ell_k = \ell_v$. □

5.2.3 Good Transcripts

We upper bounded the occurrence of a distinguishing event in the ideal world in the previous section. In this section it remains to show that in the absence of a distinguishing event, the ideal construction is indistinguishable from the real construction:

Lemma 4. *For any good transcript $\tau \in \mathsf{GoodT}$, it holds that*

$$\Pr\left[\Theta_{real} = \tau\right] = \Pr\left[\Theta_{ideal} = \tau\right].$$

Proof. In either world, the permutation P and the secret key k are sampled uniformly at random and are therefore trivially indistinguishable.

We still need to argue about the answers to the E- and the F-queries. We define $\mathcal{A}$ as the set of all α^r-values and $\mathcal{F}$ as the set of all F-query inputs that are contained in the transcript τ. Formally $\mathcal{A}$ is defined as:

$$\mathcal{A} := \{\pi^r\left(P\langle x, k^{\mathsf{pre}} \,|\, k\rangle\right) \mid (x, r) \text{ is an } E\text{-query}\}$$

As collisions between E- and F-queries do not occur in the good transcripts, we know that $\mathcal{A} \cap \mathcal{F} = \emptyset$. Therefore we can conclude that for all $\alpha^r \in \mathcal{A}$, for all $y \in \mathcal{F}$ and for all $z, z' \in \{0,1\}$:

$$\Pr_{real}[F(\alpha^r) = z, F(y) = z'] = \Pr_{ideal}[E(x, r) = z, F(y) = z'].$$

The above holds as all z, z' are independent random variables sampled uniformly from $\{0,1\}$. In particular, in either world, we evaluate F and E on different inputs only. The outputs are then sampled by a random function. □

5.2.4 Security Bounds

We obtain Lemma 2 from Lemma 1, Lemma 3 and Lemma 4. Ultimately we could verify that the CIVK construction reaches a security level equal to the key length ℓ_k.

6 Design Specification of DRACO

The design of Draco is similar to that of Lizard [HKM17b], which was in turn inspired by the Grain family [HJMM08] of stream ciphers. In particular, the inner state of Draco is distributed over two interconnected feedback shift registers (FSRs) as depicted in Fig. 1.

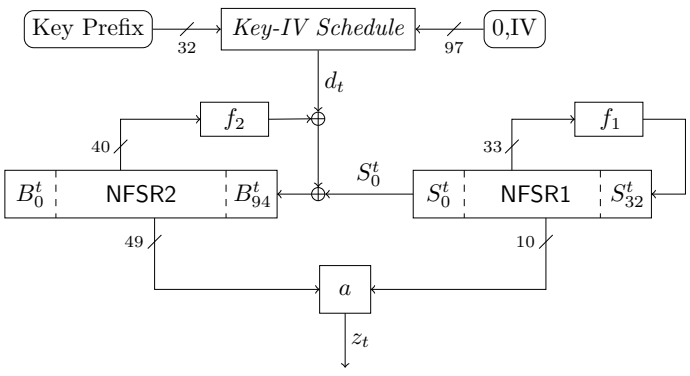

Figure 1: Draco in keystream generation mode.

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, Draco

(like Lizard) uses two NFSRs of different lengths instead. The reasons for this design choice will be explained in Section 7. 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. A major difference to Grain (and also Lizard) is that Draco continuously uses the IV and (parts of) the key not only during state initialization but also during keystream generation.

In the following, we describe the components of the cipher in detail (Sec. 6.1) and specify how it is operated during state initialization (Sec. 6.2) and keystream generation (Sec. 6.3). For the sake of clarity, subsections 6.1–6.3 contain only the technical aspects of Draco. Explanations of important design choices for our construction are given separately in Section 7, along with a discussion of the security properties of the particular components (e.g., the algebraic properties of the feedback functions).

6.1 Components

Let $K = (K_0, \ldots, K_{127})$ denote the 128-bit secret key and $IV = (IV_0, \ldots, IV_{95})$ the 96-bit public IV. The 128-bit volatile inner state of Draco is distributed over two NFSRs, NFSR1 and NFSR2, whose contents at time $t = 0, 1, \ldots$ we denote by $(S_0^t, \ldots, S_{32}^t)$ and $(B_0^t, \ldots, B_{94}^t)$, respectively (cf. Fig. 1). As NFSR1 and NFSR2 are Fibonacci-type, for $t \in \mathbb{N}$ it holds that $S_i^{t+1} := S_{i+1}^t$, $i = 0, \ldots, 31$, and $B_j^{t+1} := B_{j+1}^t$, $j = 0, \ldots, 93$.

6.1.1 Non-volatile State

Besides the 128-bit volatile inner state, Draco additionally employs a 128-bit non-volatile inner state, which is formed by the 96-bit public IV and the first 32 bits (i.e., $K_0, \ldots, K_{31}$) of the 128-bit secret key. Based on this 128-bit non-volatile inner state and the public 1-bit constant 0, in clock cycle t the *key-IV-schedule bit* (KIS bit) d_t is computed as

$$d_t := \begin{cases} x_{t \bmod 97}, & \text{for } 0 \le t \le 255, \\ K_{t \bmod 32} \oplus x_{t \bmod 97}, & \text{for } t \ge 256, \end{cases}$$

where $x_0 := 0$ and $x_i := IV_{i-1}$ for $i = 1, \ldots, 96$. The KIS-bit d_t is fed to NFSR2 as depicted in Fig. 1 and described more formally below.

6.1.2 NFSR1

Draco's NFSR1 replaces the LFSR of the Grain family of stream ciphers. NFSR1 is 33 bits wide and corresponds (with a slight adaption, see below) to the NFSR A_{12} of the eSTREAM Phase 2 (hardware portfolio) candidate ACHTERBAHN-128/80 [GGK06]. For *all* starting states, it has a guaranteed period of 2^{33} (i.e., truly maximal) and can be specified by the following update relation (during keystream generation), defining f_1 depicted in Fig. 1:

$$\begin{aligned} S_{32}^{t+1} := {} & S_0^t \oplus S_2^t \oplus S_7^t \oplus S_9^t \oplus S_{10}^t \oplus S_{15}^t \oplus S_{23}^t \oplus S_{25}^t \oplus S_{30}^t \oplus S_8^t S_{15}^t \oplus S_{12}^t S_{16}^t \\ & \oplus S_{13}^t S_{15}^t \oplus S_{13}^t S_{25}^t \oplus S_1^t S_8^t S_{14}^t \oplus S_1^t S_8^t S_{18}^t \oplus S_8^t S_{12}^t S_{16}^t \oplus S_8^t S_{14}^t S_{18}^t \\ & \oplus S_8^t S_{15}^t S_{16}^t \oplus S_8^t S_{15}^t S_{17}^t \oplus S_{15}^t S_{17}^t S_{24}^t \oplus S_1^t S_8^t S_{14}^t S_{17}^t \oplus S_1^t S_8^t S_{17}^t S_{18}^t \\ & \oplus S_1^t S_{14}^t S_{17}^t S_{24}^t \oplus S_1^t S_{17}^t S_{18}^t S_{24}^t \oplus S_8^t S_{12}^t S_{16}^t S_{17}^t \oplus S_8^t S_{14}^t S_{17}^t S_{18}^t \\ & \oplus S_8^t S_{15}^t S_{16}^t S_{17}^t \oplus S_{12}^t S_{16}^t S_{17}^t S_{24}^t \oplus S_{14}^t S_{17}^t S_{18}^t S_{24}^t \oplus S_{15}^t S_{16}^t S_{17}^t S_{24}^t \\ & \oplus \neg S_1^t \neg S_2^t \cdots \neg S_{30}^t \neg S_{31}^t \neg S_{32}^t. \end{aligned}$$

Note that NFSR1 of Draco differs from NFSR A_{12} of ACHTERBAHN-128/80 only in the additional final term $\oplus \neg S_1^t \neg S_2^t \cdots \neg S_{30}^t \neg S_{31}^t \neg S_{32}^t$, which turns the period $2^{33} - 1$ of A_{12} into the truly maximal period 2^{33} for NFSR1. That is, NFSR1 of Draco cannot get stuck in the all-zero state.

6.1.3 NFSR2

NFSR2 is 95 bits wide and uses a modified version of g from Grain-128a [ÅHJM11] as its feedback polynomial. More precisely, f_2 of NFSR2 (cf. Fig. 1) shifts six taps of g by two positions to the left in order to fit a 95 bit register (i.e., tap shifts $86 \leftarrow 88$, $89 \leftarrow 91$, $90 \leftarrow 92$, $91 \leftarrow 93$, $93 \leftarrow 95$, $94 \leftarrow 96$). Moreover, four quadratic monomials and one degree-three monomial are added to further strengthen Draco, inter alia, against algebraic attacks. This results in the following update relation (during keystream generation):

$$\begin{aligned} B_{94}^{t+1} := {} & S_0^t \oplus d_t \oplus B_0^t \oplus B_{26}^t \oplus B_{56}^t \oplus B_{89}^t \oplus B_{94}^t \oplus B_3^t B_{67}^t \oplus B_{11}^t B_{13}^t \\ & \oplus B_{17}^t B_{18}^t \oplus B_{27}^t B_{59}^t \oplus B_{36}^t B_{39}^t \oplus B_{40}^t B_{48}^t \oplus B_{50}^t B_{79}^t \oplus B_{54}^t B_{71}^t \\ & \oplus B_{58}^t B_{63}^t \oplus B_{61}^t B_{65}^t \oplus B_{68}^t B_{84}^t \oplus B_8^t B_{46}^t B_{87}^t \oplus B_{22}^t B_{24}^t B_{25}^t \\ & \oplus B_{70}^t B_{78}^t B_{82}^t \oplus B_{86}^t B_{90}^t B_{91}^t B_{93}^t. \end{aligned}$$

Note that this update relation for B_{94}^{t+1} additionally contains the masking bit S_0^t from NFSR1 (analogously to the Grain family) as well as the KIS bit d_t (unlike the Grain family).

6.1.4 Output Function a

$a : \{0,1\}^{59} \longrightarrow \{0,1\}$ builds on the construction scheme introduced in [MJSC16] as part of the FLIP family of stream ciphers (see Sec. 7 for details). For the sake of clarity, we define a through the output bit z_t of Draco at time t, which is computed as

$$z_t := \mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t^{(1)} \oplus \mathcal{T}_t^{(2)} \oplus \mathcal{T}_t^{(3)},$$

where

$$\begin{aligned} \mathcal{L}_t &:= B_7^t \oplus B_{15}^t \oplus B_{32}^t \oplus B_{47}^t \oplus B_{66}^t \oplus B_{80}^t \oplus B_{92}^t, \\ \mathcal{Q}_t &:= B_5^t B_{85}^t \oplus B_{12}^t B_{74}^t \oplus B_{20}^t B_{69}^t \oplus B_{34}^t B_{57}^t, \\ \mathcal{T}_t^{(1)} &:= B_{53}^t \oplus B_{38}^t B_{44}^t \oplus B_{23}^t B_{49}^t B_{83}^t \oplus B_6^t B_{33}^t B_{51}^t B_{73}^t \\ &\quad \oplus B_4^t B_{29}^t B_{43}^t B_{60}^t B_{81}^t \oplus B_9^t B_{14}^t B_{35}^t B_{42}^t B_{55}^t B_{77}^t \\ &\quad \oplus B_1^t B_{16}^t B_{28}^t B_{45}^t B_{64}^t B_{75}^t B_{88}^t, \\ \mathcal{T}_t^{(2)} &:= S_{26}^t \oplus S_5^t S_{19}^t \oplus S_{11}^t S_{22}^t S_{31}^t, \\ \mathcal{T}_t^{(3)} &:= B_{76}^t \oplus S_3^t B_{10}^t \oplus S_{20}^t B_{21}^t B_{30}^t \oplus S_6^t S_{29}^t B_{62}^t B_{72}^t. \end{aligned}$$

6.2 State Initialization

The state initialization process can be divided into 2 phases and is similar to the classical Grain-type initialization.

6.2.1 Phase 1: Key Loading

The registers of the keystream generator are initialized as follows:

$$\begin{aligned} B_j^0 &:= \begin{cases} K_j \oplus 1, & \text{for } j = 0, \\ K_j, & \text{for } j \in \{1, \ldots, 94\}, \end{cases} \\ S_i^0 &:= K_{i+95}, \qquad \text{for } i \in \{0, \ldots, 32\}. \end{aligned}$$

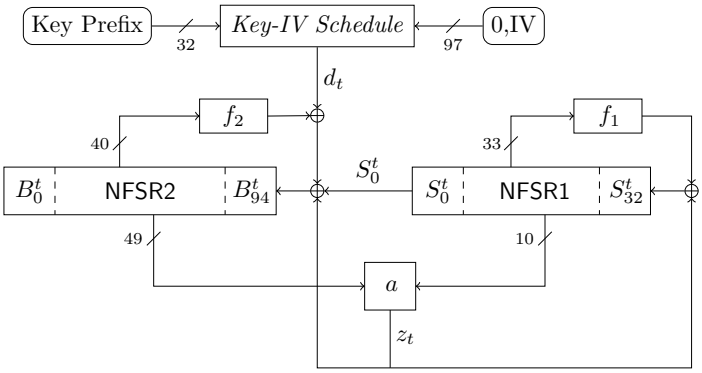

Figure 2: Draco in phase 2 of the state initialization.

6.2.2 Phase 2: Grain-like Mixing

Clock the cipher 512 times without producing actual keystream. Instead, at time $t = 0, \ldots, 511$, 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 NFSR2 and NFSR1 in phase 2. For $t = 0, \ldots, 511$, compute

$$
\begin{aligned}
B_j^{t+1} &:= B_{j+1}^t \quad \text{for } j \in \{0, \ldots, 93\}\,, \\
B_{94}^{t+1} &:= z_t \oplus S_0^t \oplus d_t \oplus B_0^t \oplus B_{26}^t \oplus B_{56}^t \oplus B_{89}^t \oplus B_{94}^t \oplus B_3^t B_{67}^t \oplus B_{11}^t B_{13}^t \\
&\quad \oplus B_{17}^t B_{18}^t \oplus B_{27}^t B_{59}^t \oplus B_{36}^t B_{39}^t \oplus B_{40}^t B_{48}^t \oplus B_{50}^t B_{79}^t \oplus B_{54}^t B_{71}^t \\
&\quad \oplus B_{58}^t B_{63}^t \oplus B_{61}^t B_{65}^t \oplus B_{68}^t B_{84}^t \oplus B_8^t B_{46}^t B_{87}^t \oplus B_{22}^t B_{24}^t B_{25}^t \\
&\quad \oplus B_{70}^t B_{78}^t B_{82}^t \oplus B_{86}^t B_{90}^t B_{91}^t B_{93}^t, \\
S_i^{t+1} &:= S_{i+1}^t \quad \text{for } i \in \{0, \ldots, 31\}\,, \\
S_{32}^{t+1} &:= z_t \oplus S_0^t \oplus S_2^t \oplus S_7^t \oplus S_9^t \oplus S_{10}^t \oplus S_{15}^t \oplus S_{23}^t \oplus S_{25}^t \oplus S_{30}^t \oplus S_8^t S_{15}^t \oplus S_{12}^t S_{16}^t \\
&\quad \oplus S_{13}^t S_{15}^t \oplus S_{13}^t S_{25}^t \oplus S_1^t S_8^t S_{14}^t \oplus S_1^t S_8^t S_{18}^t \oplus S_8^t S_{12}^t S_{16}^t \oplus S_8^t S_{14}^t S_{18}^t \\
&\quad \oplus S_8^t S_{15}^t S_{16}^t \oplus S_8^t S_{15}^t S_{17}^t \oplus S_{15}^t S_{17}^t S_{24}^t \oplus S_1^t S_8^t S_{14}^t S_{17}^t \oplus S_1^t S_8^t S_{17}^t S_{18}^t \\
&\quad \oplus S_1^t S_{14}^t S_{17}^t S_{24}^t \oplus S_1^t S_{17}^t S_{18}^t S_{24}^t \oplus S_8^t S_{12}^t S_{16}^t S_{17}^t \oplus S_8^t S_{14}^t S_{17}^t S_{18}^t \\
&\quad \oplus S_8^t S_{15}^t S_{16}^t S_{17}^t \oplus S_{12}^t S_{16}^t S_{17}^t S_{24}^t \oplus S_{14}^t S_{17}^t S_{18}^t S_{24}^t \oplus S_{15}^t S_{16}^t S_{17}^t S_{24}^t \\
&\quad \oplus \neg S_1^t \neg S_2^t \cdots \neg S_{30}^t \neg S_{31}^t \neg S_{32}^t,
\end{aligned}
$$

where z_t and d_t are computed as described in Subsection 6.1.4.

6.3 Keystream Generation

At the end of the state initialization, the 33-bit (initial) state of NFSR1 is $\left(S_0^{512}, \ldots, S_{32}^{512}\right)$ and the 95-bit (initial) state of NFSR2 is $\left(B_0^{512}, \ldots, B_{94}^{512}\right)$. The first keystream bit that is used for plaintext encryption is z_{512}. For $t \geq 512$, the states $\left(S_0^{t+1}, \ldots, S_{32}^{t+1}\right)$ and $\left(B_0^{t+1}, \ldots, B_{94}^{t+1}\right)$ and the output bit z_t are computed using the relations given in Subsection 6.1. Fig. 1 depicts the structure of Draco during keystream generation.

As Draco is designed to be operated in packet mode, the maximum size of a plaintext packet encrypted under the same key/IV pair is 2^{32} 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 $z_{512}, z_{513}, \ldots$ 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 $y_i := x_i \oplus z_{i+512}$, $i = 0, \ldots, |X| - 1$. Decryption (given that the secret session key and the public IV are known) works analogously.

Note that, though we use the terms plaintext/ciphertext packet here, Draco is really a (synchronous) stream cipher. I.e., the keystream bits $z_{512}, z_{513}, \ldots$ 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 ciphertext bits y_i.

7 Design Considerations

In this section, we provide additional explanations w.r.t. our design, which were omitted in Section 6 for the sake of clarity. As Draco has a Grain-like structure, we particularly focus on respective similarities and differences. Based on several of the following properties, we will then argue in Section 8 why we believe that Draco resists the currently known types of attacks against stream ciphers.

7.1 The Key-IV Schedule

The Key-IV Schedule defined in Subsection 6.1 is designed in such a way that for all $t \geq 0$ the mapping $D_t : \{0,1\}^{32} \times \{0,1\}^{96} \longrightarrow \{0,1\}^{128}$, defined as

$$D_t(K_0, \ldots, K_{31}, IV_0, \ldots, IV_{95}) := (d_t, d_{t+1}, \ldots, d_{t+127})$$

is a bijective GF(2)-linear mapping. This follows directly from Lemma 5.

Lemma 5. *Fix some $r \in \{0, \ldots, 96\}$. The system of GF(2)-linear equations in the variables $u_0, \ldots, u_{31}, v_0, \ldots, v_{96}$*

$$v_r = 0 \tag{1}$$
$$u_0 \oplus v_0 = u_1 \oplus v_1 = \cdots = u_{31} \oplus v_{31} = 0 \tag{2}$$
$$u_0 \oplus v_{32} = \cdots = u_{31} \oplus v_{63} = 0 \tag{3}$$
$$u_0 \oplus v_{64} = \cdots = u_{31} \oplus v_{95} = 0 \tag{4}$$
$$u_0 \oplus v_{96} = u_1 \oplus v_0 = \cdots = u_{31} \oplus v_{30} = 0. \tag{5}$$

has only one solution: $u_0 = \cdots = u_{31} = v_0 = \cdots = v_{96} = 0$.

This system corresponds the situation that t is chosen in such a way that the constant 0 is added at position $t+r$, which is the case if $r \equiv 97 - t \mod 97$. Moreover, $(u_0, \ldots, u_{31})$ is obtained from $(K_0, \ldots, K_{31})$ by some cyclic shift, and $(v_0, \ldots, v_{96})$ is obtained from $(x_0, \ldots, x_{96})$ by a cyclic right shift by r positions.

Proof. Note that $u_s = 0$ for $s = r \mod 32$ follows from $v_r = 0$. This implies $v_{s-1} = 0$ (by (5)) and $u_{s-1} = 0$ (by (2)) and $v_{s-2} = 0$ (by (5)) and $u_{s-2} = 0$ (by (2)) and so on, i.e., $u_s = u_{s-1} = \cdots = u_0 = 0$. On the other hand, $v_s = 0$ (by (2)) which implies $u_{s+1} = 0$ (by (5)) which implies $v_{s+1} = 0$ (by (2)) and so on, i.e., $u_s = u_{s+1} = \cdots = u_{31} = 0$. As all u-bits are zero, by (2-5), all v-bits have to be zero, too. □

Note that the bijectivity of D_t makes Draco immune against the following type of chosen-IV TMDTO-attacks, which we call Zero d-stream attacks.

7.1.1 Zero d-stream Attacks

Let $\mathcal{K}(0) \subseteq \{0,1\}^{32}$ be the set of all key prefixes k^{pre} for which there is some initial value $IV \in \{0,1\}^{96}$ such that $d_t(k^{\mathsf{pre}}, IV) = 0$ for all $t \geq 0$. Correspondingly, let $\mathcal{IV}(0) \subseteq \{0,1\}^{96}$ be the set of all initial value $IV \in \{0,1\}^{96}$ for which there is a key prefix $k^{\mathsf{pre}} \in \{0,1\}^{32}$ such that $d_t(k^{\mathsf{pre}}, IV) = 0$ for all $t \geq 0$. For all volatile internal states $S \in \{0,1\}^{128}$ let

$Z_{\vec{0}}(S)$ denote the block of the first 128 keystream bits generated on S under the condition that the corresponding 128 d-stream bits are all zero.

Offline

1. The attacker computes $\mathcal{K}(0)$ and $\mathcal{IV}(0)$

2. The attacker computes $Z_{\vec{0}}(S)$ for 2^{96} randomly and mutually independently chosen internal states $S \in \{0,1\}^{128}$ and stores them in a database $\mathcal{D}$.

Online

3. The attacker asks for the keystream packet $P(k^*, IV) \in \{0,1\}^{32}$, $k^* \in \{0,1\}^{128}$ the secret key, for all initial values $IV \in \mathcal{IV}(0)$ and checks if $P(k^*, IV) \in \{0,1\}^{32}$ contains a sub block D of length 128 which occurs in $\mathcal{D}$.

4. In the case that a collision $D = Z_{\vec{0}}(S)$ was found, compute k^* from S.

Note that if $k^* \in \mathcal{K}(0)$ then, in step 3, the attacker knows the packet $P(k^*, IV^*)$ for some $IV^* \in \mathcal{IV}(0)$. This packet contains 2^{32} keystream blocks $Z_{\vec{0}}(S')$ for the internal state $S' \in \{0,1\}^{128}$. By the birthday paradox, there will be a collision with $\mathcal{D}$ which yields the secret key. If $k^* \notin \mathcal{K}(0)$ then the attack fails. Consequently, the success probability of the attack is around $|\mathcal{K}(0)| \cdot 2^{-32}$, while the cost is at least 2^{96}.

Due to the fact that D_t is bijective for all $t \geq 0$ (see Subsection 7.1) we have that $|\mathcal{K}(0)| = |\mathcal{IV}(0)| = 1$, which implies that the Zero d-stream attack against Draco does not beat exhaustive key search.

7.2 NFSR1

NFSR1 is 33 bits wide and replaces the maximum-length LFSR of the Grain family. Especially in view of the halved size of the cipher's volatile inner state (128 bits for Draco vs. 256 bits for Grain-128a), employing an NFSR is favorable here in order to strengthen the design against algebraic and fast correlation attacks such as [TIM+18]. Unfortunately, not much is known yet about how to generically construct large, cryptographically strong NFSRs with maximal period. For FSR sizes up to 30–40 bits, however, corresponding non-linear feedback functions can be found experimentally. In [GGK06], the designers of the eSTREAM Phase 2 (hardware portfolio) candidate ACHTERBAHN-128/80 provide a list of 13 such NFSRs, ranging in size from 21 bits (NFSR A_0) to 33 bits (NFSR A_{12}). The latter is perfectly sufficient for our design as, due to the restriction to packet mode with a maximum of 2^{32} keystream bits per key/IV pair, Draco actually does not need as large guaranteed periods as the Grain family.

While in the original Grain family, an all-zero LFSR initial state is tolerable due to the FSRs' sizes (both of key length), this must strictly be avoided for *small-state* Grain-like stream ciphers. More precisely, for one in 2^{33} key-IV combinations, the Grain-like mixing employed by Draco during state initialization will result in an all-zero state of NFSR1. In this situation, NFSR A_{12} of ACHTERBAHN would be stuck in the all zero-state, because (like a maximum-length LFSR of this size) it 'only' has period $2^{33} - 1$ for *non-zero* starting states. On contrast, Draco's NFSR1 yields truly maximal period 2^{33} for *any* starting state. This is achieved by adding the term $\oplus \neg S_1^t \neg S_2^t \cdots \neg S_{31}^t \neg S_{32}^t$ to ACHTERBAHN's NFSR A_{12} as described in Section 6.1, thus simply 'gluing' the all-zero state into the original state cycle of NFSR A_{12} between the states $(1, 0, \ldots, 0)$ and $(0, \ldots, 0, 1)$.

Accordingly, we can assess the security of our NFSR1 on the basis of the following properties of A_{12} given in [GGK06]: a nonlinearity of 114688, an order of correlation

immunity of 6, and a diffusion parameter[5] of 54. Moreover, it is easy to see that A_{12}'s feedback function is balanced and, thus (as it is 6th order correlation-immune), 6-resilient. For comparison, in Grain-128a the feedback function of the LFSR (which is replaced by NFSR1 in Draco) has resiliency 5. Finally, the algebraic degree of A_{12}'s feedback function is 4.

A major reason for choosing A_{12} from ACHTERBAHN as a basis for Draco's NFSR1 is also its hardware efficiency. Despite a comparatively large algebraic normal form, the designers of ACHTERBAHN are able provide a compact hardware realization of A_{12}'s feedback function consuming only 31.75 gate equivalents (GE) and having logical depth three (see Sec. 9 for further details w.r.t. hardware complexity and an explanation of corresponding units of measure like GE).

7.3 NFSR2

NFSR2 is 95 bits wide and its feedback polynomial is a modified version of g from Grain-128a [ÅHJM11]. In contrast to NFSR1, the period of NFSR2 during keystream generation is unknown because even after state initialization, it is not operated in a self-contained manner. More precisely, due to the masking bit from NFSR1 and the KIS bit d_t, NFSR2 is actually a filter instead of a real NFSR (cf. corresponding remark for the Grain family in [HJMM08]).

As described in Section 6.1, for f_2 of Draco we shifted six taps of g from Grain-128a by two positions to the left in such a way that the property of g that no tap appears more than once is preserved in f_2. Moreover, we added the following monomials to further strengthen Draco in the face of its smaller volatile inner state: $B^t_{36}B^t_{39}$, $B^t_{50}B^t_{79}$, $B^t_{54}B^t_{71}$, $B^t_{58}B^t_{63}$, $B^t_8B^t_{46}B^t_{87}$. Note that the set of tap indices of these new nonlinear monomials is disjoint from the set of tap indices of all other monomials of f_2. In consequence, several important properties of g from Grain-128a like its balancedness and its resiliency of 4 carry over to f_2. Similarly, the security of f_2 of Draco w.r.t. linear approximations can hence be lower boundend by that of Grain-128a (2^{14} best linear approximations with bias $63 \cdot 2^{-15}$). As proven in [MJSC16], the aforementioned disjointness property w.r.t. the newly added taps implies that the nonlinearity of f_2 of Draco can be computed on the basis of the nonlinearity of g from Grain-128a (267403264) and its number of variables (29) together with the nonlinearity of the sum of the new monomials (976) and its number of variables (11) as follows: $2^{11} \cdot 267403264 + 2^{29} \cdot 976 - 2 \cdot 267403264 \cdot 976 = 549656723456$ ($\approx 2^{39}$). As for g from Grain-128a, the algebraic degree of f_2 of Draco is 4.

Beyond these properties, g has so far successfully thwarted all cube-like attacks against the initialization of Grain-128a. In Section 8.3, we derive corresponding security arguments for Draco.

7.4 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 volatile inner state of Draco is smaller than that of Grain-128a, we decided that the output function should have more inputs and larger algebraic degree instead. It builds on the construction scheme introduced in [MJSC16] as part of the FLIP family of stream ciphers. More precisely, Draco's output function a 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, a triangular function with three

[5] The diffusion parameter λ was determined experimentally in [GGK06] 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).

monomials, and another triangular function with four monomials, where each tap of NFSR1 and NFSR2 appears at most once in a.

As a consequence, the output function of Draco is defined over 59 variables, balanced, and has, according to lemmata 3–6 in [MJSC16], the following security properties: a nonlinearity of 287580136809693184 ($\approx 2^{57}$), a resiliency of 9, 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 44 variables and 'gracefully degrades' into the direct sum of a linear function with eight or nine (depending on the value S_3^t) monomials, a quadratic function with four to six (depending on the values S_{20}^t and $S_6^t S_{29}^t$) monomials, and a triangular function with seven monomials, which again conforms to the construction principle introduced in [MJSC16] and leads to the following worst-case security properties for that situation: a nonlinearity of 8634823344128 ($\approx 2^{42}$), a resiliency of 8, 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. [HJM06, ÅHJM11, HJMM08]) 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 [DS11] on Grain-128, which lead to a change of tap positions in the output function of Grain-128a [ÅHJM11]).

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 [Gol96] to assess the security of nonlinear filter generators consisting of a single LFSR and a nonlinear output function. Note that, e.g., the fast correlation attacks against Grain v1 and Grain-128a presented in [TIM+18] actually match this cipher model as they treat Grain's NFSR as (part of) a filter for its LFSR. For further details about Golić's findings and how they influenced Draco's output function, we refer the reader to Appendix B.

Another cryptanalytic result motivating our selection of tap positions for the output function of Draco 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 [Kra02] and applied to Grain-128 by Stegemann in [Ste07], is that (roughly speaking) the distance between the lowest and the highest tap index of a monomial should be large for as many monomials as possible (see Section 8.6 for further details).

7.5 Continuous Key and IV Usage

In this subsection, we provide further details about how the generic CIVK construction introduced in Subsection 3.2 is instantiated concretely through Draco. In particular, we argue about the choice of the different parameter lengths. With Draco we want to achieve a security level of 128 bits. Accordingly, the key length is chosen to be 128 bits.

As Draco is designed to operate in packet mode and a key/IV pair may be used to generate at most 2^{32} keystream bits, the key prefix length is $\log_2 2^{32} = 32$ bits. This also determines the length of the initial value: As the desired security level is 2^{128} and the packet length is 2^{32} bits, we need to set the length of the initial value to be $128 - 32 = 96$ bits. This corresponds to a total volatile internal state size of 128 bits and a total non-volatile internal state size of 128 bits.

It is easy to see that Draco's mixing phase in fact realizes a bijection (between the 256-bit internal states at $t = 0$ and those at $t = 512$) as assumed by our security proof for the CIVK construction given in Section 5. Draco deviates from the generic construction scheme only in a tiny detail. During key loading at $t = 0$, the first key bit is inverted (i.e., $B_0^0 := K_0 \oplus 1$). This is to avoid the "sliding property" of Grain v1 and Grain-128 that was pointed out in [DCKP08] (see Section 8.4 for further details). In terms of our TMDTO security proof, however, this modification is completely irrelevant.

7.6 Packet Mode

Stream ciphers have a long history when it comes to protecting digital communication. In 1987, Rivest designed RC4 [Sch95], which was later used in SSL/TLS [DR08] and the wireless network security protocols WEP [Ins97] and TKIP (often called WPA) [Ins04]. Other well-known stream cipher examples are E_0 of the Bluetooth standard [SIG14] and A5/1 of GSM [BGW99]. Unfortunately, E_0 and A5/1 have been shown to be highly insecure (see, e.g., [LMV05] and [BB06]) and RC4 also shows severe vulnerabilities, which led to its removal from the TLS protocol [Pop15] and rendered other protocols like WEP insecure [FMS01]. In 2004, the eSTREAM project [ECR08] 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 [HJM06], MICKEY 2.0 [BD06] and Trivium [CP05].

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. 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. 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 802.11-2020 standard [Ins21] implies that at most 11454 bytes (i.e., $< 2^{17}$ bits) are encrypted under the same key/IV pair using CCMP. 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 1.3 [Res18], the maximum amount of data encrypted under the same key/IV pair is $2^{14} + 2^8$ bytes (i.e., $2^{17} + 2^{11} < 2^{18}$ bits), as long as RC4, which is now forbidden for all TLS versions by RFC 7465 [Pop15], is not used.

Considering the above examples, Draco's maximum packet length of 2^{32} bits should be more than sufficient for most communication scenarios in the foreseeable future.

8 Cryptanalysis

In the following subsections, we will argue for several types of attacks which weakened or even broke other stream ciphers in the past, why we believe that Draco will resist them. In Section 4 we already argued about time-memory-data tradeoff attacks against the CIVK construction that underlies Draco.

The discussion in this section will build on a variety of results that illuminate the security of Grain-v1 against these attacks, and that address a number of established security parameters. Since Grain and Draco are very similar in the structural elements relevant to algebraic and correlation attacks, it seems sufficient to us in the given context to describe the extent to which Draco meets or exceeds the design criteria relevant to Grain's security. Of course, this does not mean provable security against every conceivable variant of such attacks. But establishing a more extended and detailed formal framework for formally proving the security of grain-like ciphers would have to go beyond the existing state of the art and would thus be a challenging scientific project in its own right, clearly beyond the scope of this paper. We think that further development of this formal framework should come from new attacks ideas being published from within the scientific community.

8.1 Correlation Attacks, Linear Approximations

Correlation and fast correlation attacks like those in [MJSC16, TIM+18, ZGM17, TMA20, WLLM19] are a major threat to Grain-like stream ciphers. They target the cipher's LFSR and are based on finding sufficiently biased linear approximations of the NFSR's feedback as well as of the output function. In Draco, Grain's LFSR is replaced by NFSR1 with high nonlinearity and correlation immunity. Moreover, the output function has much more inputs, a higher resiliency and a much higher nonlinearity than that of Grain-128a. Furthermore, the feedback function of NFSR2 was additionally hardened as explained in Section 7.3.

In [MJSC16], Méaux et al. point out the importance of "good balancedness, non-linearity and resiliency properties" of the filtering function in order to withstand correlation attacks [Sie85] and fast correlation attacks [MS89]. As explained in Section 7.4, Draco features a rather heavy output function to compensate for the smaller volatile part of the inner state compared to the original Grain family. It is defined over 59 variables and has nonlinearity of about 2^{57}, whereas the output function of Grain-128a is defined over 17 variables and has nonlinearity 61440. Moreover, the resiliency of Draco's output function is 9 compared to 7 for that of Grain-128a.

In [BGM06], 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. As possible countermeasures, Berbain, Gilbert and Maximov proposed the following modifications [BGM06]: "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". 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 order to obtain a small bias." As a consequence, g was chosen such that it has nonlinearity 267403264 ($\approx 2^{28}$) and resiliency 4.

As explained in Section 7.3, the feedback function f_2 of NFSR2 in Draco builds on that of Grain-128a in a way that preserves its balancedness and resiliency, but features an even higher nonlinearity ($\approx 2^{39}$). Moreover, in accordance with the above suggestions from [BGM06] and the construction principle underlying g of Grain-128a (see previous paragraph), the output function of Draco has more than three times as many inputs, a much higher nonlinearity and a higher resiliency than that of Grain-128a (cf. values at the beginning of this subsection) in order to strengthen it against linear approximations. In particular, it is defined over the state variables of both FSRs, featuring monomials of all degrees between one and seven defined over NFSR2 (cf. triangular function $\mathcal{T}_t^{(1)}$ in Section 6.1.4), monomials of degrees one, two, and three over NFSR1 (cf. triangular

function $\mathcal{T}_t^{(2)}$), and monomials of degrees two, three, and four with variables from both FSRs mixed (cf. triangular function $\mathcal{T}_t^{(3)}$).

Also note that (fast) correlation attacks against Grain-like structures as published in [MJSC16, TIM+18, ZGM17, TMA20, WLLM19] target the ciphers' LFSR. However, as described in Section 7.2, the Grain family's LFSR is replaced by an NFSR with high nonlinearity and correlation immunity for Draco. This approach was already employed for Lizard, which is the only Grain-like stream cipher completely unaffected by (fast) correlation attacks, so far.

In [TMA20], Todo, Meier and Aoki study the data limitation of small-state stream ciphers in the context of correlation attacks. For Plantlet, which targets 80-bit security, they can recover the secret key if about 2^{53} keystream bits per key/IV combination are available. Fortunately, this condition cannot be met in practice as the cipher's designers set a corresponding limit of 2^{30} bits, which is considered "conservative" by Todo, Meier and Aoki. In line with these findings, Draco (whose key and and state size are larger than those of Plantlet) has a maximum packet length of 2^{32} bits.

Finally, as explained in Section 7.4, the choice of tap positions for Draco's output function follows the concept of (full) positive difference sets, which was introduced by Golić in [Gol96] as a design criterion to strengthen nonlinear filter generators against correlation attacks.

8.2 Algebraic Attacks

For algebraic attacks against Draco, one has to differentiate between two basic approaches. First, an attacker could express observed keystream bits as functions of the unknown 128 key bits and then try to solve the corresponding system of equations. This, however, 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 the following second attack approach: expressing observed keystream bits as functions of the unknown 128 bits of the volatile initial state at $t = 512$ and the unknown 32-bit key prefix (used continuously during keystream generation) and then trying to solve the corresponding system of equations. Consequently, for the remainder of this subsection, we will focus on the second approach.

First of all, note that, to the best of our knowledge, no successful (i.e., having complexity lower than 2^{128}) algebraic attacks that can recover arbitrary initial states for Grain-128a have been reported so far.[6] Due to the smaller volatile inner state of Draco, the number of variables of the 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-128a. As pointed out in Section 7.4, Draco's output function builds on the construction scheme introduced in [MJSC16], depends on 59 variables, has nonlinearity of about 2^{57}, algebraic immunity of at least 7 and 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. Furthermore, we hardened NFSR2 against algebraic attacks by adding five more nonlinear monomials as compared to g in Grain-128a (cf. Sec. 7.3). Based on these properties, we expect that algebraic attacks against full Draco will not have complexity lower than that of exhaustive key search.

[6] 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 [BGJ09]. 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 Draco due to the maximum packet length of 2^{32} bits.

Also note that even when guessing the shorter NFSR1, Draco's output function still depends on at last 44 variables and has, inter alia, the following worst-case security properties: nonlinearity of about 2^{42}, algebraic immunity at least 7, and fast algebraic immunity at least 8. For comparison, the full output function of Grain-128a depends on 17 variables, has nonlinearity 61440 and algebraic degree 3. Thus, in the context of a corresponding guess-and-determine attack, an algebraic attack on NFSR2 similar to the one in [BGJ09] will have large enough complexity.

Guessing the larger NFSR2 is even less promising from an attacker's point of view. Due to its size of 95 bits and the 128-bit security level targeted by Draco, a successful state-recovery attack against the full cipher would have to subsequently recover the 33-bit inner state of NFSR1 and the 32-bit key prefix underlying the key-IV-schedule with time/memory/data complexity below $2^{128-95} = 2^{33}$. In particular, these 65 remaining unknowns also influence the further state updates of NFSR2, which, on contrast to NFSR1, is not operating autonomously during keystream generation. So while in this phase guessing NFSR1 allows to *eliminate* it (including its feedback function and its contribution to the output function) from subsequent steps of the cryptanalysis, this is not the case when guessing NFSR2. That is, an attacker guessing NFSR2 would have to recover 65 unknowns with time/memory/data complexity below 2^{33}, while still being faced with both feedback functions and the full output function of Draco.

8.3 Conditional Differentials, Cube Attacks

In [LM12], Lehmann and Meier study the security of Grain-128a against dynamic cube attacks and differential attacks. They come 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." The currently best key-recovery cube attack against round-reduced Grain-128a is presented in [TIHM17]. It is based on the division property and works for 183 initialization rounds.

Draco has 512 rounds in phase 2 of the state initialization, where the Grain-like mixing is performed as described in Section 6.2. On top of that, from the second half of phase 2 onwards (i.e, for all $t \geq 256$), the 32-bit prefix of the secret key is continuously involved in the state update of NFSR2.

Note that the volatile inner state of Draco (128 bits) is smaller than that of Grain-128a (256 bits), whereas the output function is much more dense. It depends on 59 variables as compared to 17 in Grain-128a. The output function of Draco also has more nonlinear monomials (15) than that of Grain-128a (5). Moreover, now both FSRs are nonlinear and the feedback function of NFSR1 is defined over more inputs (40 vs. 19) and has more nonlinear monomials (15 vs. 10) than that of Grain-128a's NFSR.

The combination of a smaller volatile state and more dense feedback and output functions causes a faster diffusion of differentials and of the monomial structure for Draco. Together with the doubled number of initialization rounds, this should make Draco at least as resistant against differential attacks and cube attacks as Grain-128a, which seems to be already sufficiently secure in that respect.

In 2021 Horn [Hor21] studied the resistance of an earlier version of Draco against cube attacks. Since then, the key-IV-schedule was slightly changed to prevent the zero d-stream attacks mentioned in Subsubsection 7.1.1. In particular the work by Horn considers a key prefix of length 33 instead of 32 and an IV of length 95 instead of 96 with an additional 0-prefix. This will not significantly change the results obtained in the analysis against cube attacks.

Horn [Hor21] considered only 99 and 100 initialization rounds instead of the full 512 as "the superpoly recovery for Draco frequently turned out to become computationally infeasible even for a very small number of initialization rounds." Horn observed that in each clock cycle only one IV bit enters the internal state of NFSR2. He found practical distinguishers for 99 and 100 initialization rounds. Yet, he was not successful in attacking 101 initialization rounds. The author states that "it was not possible to recover the superpoly of a cube just a few rounds after the cube variable with the highest index is introduced." Further, since Draco uses 512 initialization rounds, Horn considers the margin more than sufficient to provide very high security against his cube attacks. Horn concludes that the "extremely fast growing complexity of these superpolys of an even simplified version of Draco, again supports our assumption that Draco is extremely resistant against the considered attack."

8.4 Slide Attacks, Related Key Attacks

In [Küç06], 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 [DCKP08] 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.[7] 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." [DCKP08] As a reaction, the designers of Grain-128a changed the 22-bit constant $(1, \ldots, 1)$ that was used in the state initialization of Grain-128 to $(1, \ldots, 1, 0)$.

In Draco, no constants are used during state initialization. Instead, to avoid the above sliding property, we set $B_0^0 := K_0 \oplus 1$ in phase 1 of the state initialization (cf. Sec. 6.2). As a result, for a key/IV pair (K, IV), a related key/IV pair (K^*, IV^*) in the sense of [DCKP08] would have to satisfy

$$K_0^* = K_1 \oplus 1. \tag{6}$$

Let d_t and d_t^* denote the key-IV-schedule bits computed on the basis of (K, IV) and (K^*, IV^*), respectively, as described previously in Section 6.1. For the sliding property from [DCKP08] to occur, $d_{t+1} = d_t^*$ would need to hold for $t \geq 0$. In particular, we get

$$IV_1 = d_1 = d_0^* = IV_0^* \tag{7}$$

and

$$K_1 \oplus IV_1 = d_{289} = d_{288}^* = K_0^* \oplus IV_0^*. \tag{8}$$

It is easy to see that equations (6), (7), and (8) cannot be satisfied simultaneously.

Note that, without inverting K_0 in phase 1 of the state initialization together with having different definitions of d_t for $t \leq 255$ and $t \geq 256$, Draco would in fact suffer from a variant of the sliding property, despite continuously employing the IV and the 32-bit key prefix for state update during keystream generation.

Let us also point out that, as stated by De Cannière, Küçük and Preneel in [DCKP08] and cited above, we too consider the supposition underlying related-key attacks to be rather strong. In particular, we do not claim security for situations where a potential

[7] More precisely, this speed up refers to making the key candidate checks more efficient. The actual number of key candidate checks, however, is not reduced compared to exhaustive key search. Still, we consider such a sliding property undesirable as it might pave the way for other attacks and, hence, seek to avoid it for Draco.

victim generates keystream under secret related keys and an attacker tries to recover one or more of these. Typical motivations of this scenario would be fault attacks (e.g., an attacker manipulating the secret inner state and/or the unknown key bits via clock glitches, voltage spikes, optical or electromagnetic fault injection etc.) or the usage of a weak (session) key derivation method. The latter is obviously a blatant security flaw on its own that needs to be avoided irrespective of the employed cipher. And for protecting against fault attacks, various established countermeasures are available on the hardware design level (see, e.g., [KSV13] for an overview). Correspondingly, we do not make security claims regarding other types of side-channel attacks either.

8.5 Weak Key/IV Pairs

In [ZW09], Zhang and Wang introduce the notion of *weak key/IV pairs* for the Grain family of stream ciphers. They show that Grain-128 has 2^{96} such pairs, which lead to an all-zero initial state of the LFSR, and use them to mount distinguishing attacks and initial state recovery attacks. In [ÅHJM11], 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."

In analogy to the definition of Zhang and Wang, weak key/IV pairs for Draco would lead to an all-zero initial state of NFSR1. Such pairs, however, are now completely unproblematic (and, hence, not *weak* anymore) as the 33-bit-wide NFSR1 has truly maximal period 2^{33}. In particular, unlike the LFSR of the Grain family, it cannot get stuck in the all-zero state.

Note that without adding the term $\oplus\neg S_1^t\neg S_2^t\cdots\neg S_{30}^t\neg S_{31}^t\neg S_{32}^t$ to ACHTERBAHN's NFSR A_{12} for obtaining NFSR1 of Draco (cf. Sec. 6.1), there would have actually been about 2^{191} weak key/IV pairs out of 2^{224} total key/IV pairs, leading to a probability of 2^{-33} for using a weak pair. Thus, corresponding attacks might have posed a real threat to Draco, which is now avoided.

8.6 BDD-based Attacks

In [Kra02], 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. Stegemann later showed in [Ste07] how this approach can be transferred to NFSR-based stream ciphers like Trivium and Grain.

In contrast to TMD tradeoff attacks or correlation attacks, which potentially require a lot of known keystream, 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 state length n) is required to recover the corresponding initial state.

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 [Ste07] for further details). In the setting of Stegemann, 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 Draco, the maximum number of active monomials would be 19 for NFSR1, 6 for NFSR2 and at least 10 for the output function a. Consequently, we expect that, despite the smaller volatile inner state, Draco will also resist BDD attacks.

8.7 Preventing Banik et al. and Esgin-Kara Attacks

Banik's attack [BCI+21] against Sprout is based on the LFSR-property that the constant zero internal state occurs in Sprout with a certain probability. This state generates a stream of zeros. As Draco does not use any LFSR, this attack can not applied to Draco.

The Banik, Baroti, Isobe attack against Plantlet [BBI19] exploits Plantlet's property that pairs of internal states which differ only in position 43 generate identical keystream blocks of length 41. This property is due to the comparatively large distance between certain taps of Plantlet's LFSR. To prevent this type of attack, the Atom stream cipher uses an additional second key filter which is driven by a 7-bit LFSR. As all pairs of neighbored taps of both of Draco's NFSRs have a sufficiently small distance, the BBI-attack can not be applied to Draco. This is the reason why we decided to use only a simple cyclic filter as key schedule for Draco, and not an LFSR-driven one.

The Esgin-Kara attack against Sprout [EK15] uses the fact that the key bits coming from the key filter are multiplied by a term depending on the volatile state. These key bits can be shown to be zero in certain clock cycles. This implies that certain keystream blocks do only depend on the volatile internal states, which allows for nontrivial TMDTO-attacks. This type of attack can not be applied to Draco as the d-bits coming from the Draco key-IV schedule are linearly added to the state update function of NFSR1. Moreover, the Draco key-IV schedule has the property that each 128-bit key stream block depends on all key prefix- and IV-bits. This prevents attacks like Esgin-Kara's Subsection 3.4 of [BCI+21].

9 Hardware Results

In this section, we present the hardware results for our new stream cipher Draco and compare them to those of Atom [BCI+21] and Grain-128a [ÅHJM11][8], which, like Draco, accepts 128-bit keys and 96-bit IVs. The reasons for focusing on Grain-128a are twofold. First, it is a natural choice for comparison due to the close structural relation between Draco and the Grain family of stream ciphers as explained in sections 6 and 7. Second, and more importantly, Grain v1 (the 80-bit version of Grain-128a) turned out to be the most hardware efficient member of the eSTREAM [ECR08] portfolio (see tables 1–4 and figures 1–3 in [GB08]) and, hence, the Grain family of stream ciphers can be considered as a benchmark for new designs. Also note that Draco is the first small-state stream cipher offering full 128-bit security against key recovery *and* distinguishing attacks, which is why a comparison to, e.g., Plantlet or Lizard would not be appropriate, here.

Second we chose to compare Draco to Atom. Atom is a lightweight stream cipher that was recently published in ToSC [BCI+21]. Atom is a reasonable comparison as it also uses a 128-bit key and it further stores the secret key externally, i.e. it builds upon CKEY that we introduced earlier. We further implemented a version of Atom that stores its secret key internally in the hardware module in an additional register that is denoted in Table 2 as $\text{Atom}_{[\mathsf{K}]}$. This is done to allow the comparison to variants of Draco that store the key prefix, resp. IV, locally in the hardware module as described below. In particular, for $\text{Draco}_{[\mathsf{KI}]}$ and $\text{Atom}_{[\mathsf{K}]}$ there are no dependencies to external resources, as is the case for Grain-128a.

In line with papers like [Fel07, GB08], which evaluate candidates in the eSTREAM hardware category, we focus on application-specific integrated circuits (ASICs) with standard CMOS libraries. ASICs are the prevalent hardware component in lightweight application scenarios, such as radio frequency identification (RFID) technology, and likewise

[8] Note that Grain-128a actually comes in two flavors: *authenticated encryption* and *encryption only*. For reasons of fairness, we naturally consider the more lightweight encryption-only variant of Grain-128a in our comparison to Draco. In fact, the authentication mechanism of Grain-128a is completely independent of the underlying keystream generator and could as well be used in connection with Draco.

Table 2: Hardware metrics for Draco and Grain-128a.

Design	**Area** [GE]	**Power** [µW]				
		100 KHz	1 MHz	10 MHz	100 MHz	1 GHz
Atom	2976	67.9	71.2	104.9	441.9	3811.7
Atom$_{[K]}$	3858	88.9	92.3	126.1	463.9	3842.3
Grain-128a	2795	67.3	71.6	115.3	551.4	4912.9
Draco	2142	48.8	51.6	79.2	355.6	3119.3
Draco$_{[K]}$	2368	54.2	57.0	84.7	369.1	3134.1
Draco$_{[I]}$	2805	64.6	67.7	95.1	372.3	3144.7
Draco$_{[KI]}$	3025	69.9	72.6	100.6	377.6	3150.0

important for highspeed cryptographic processing, such as bitcoin mining. 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 [AHM14] 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). In ASIC-based highspeed processing, on the other hand, energy consumption is becoming the main cost factor (see, e.g., [DV18]).

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" [GB08]. Consequently, it is crucial to use the same design flow for all implementations that are to be compared. In Appendix C.1, we provide a detailed specification of the tools and methodology employed for deriving the hardware evaluation results summarized in Table 2. After state initialization, all implementations produce one keystream bit per clock cycle, leading to identical throughput rates at identical clock speeds.

Remember that in contrast to Grain-128a, half of Draco's 256-bit inner state is actually held constant (consisting of the 32-bit key prefix and the 96-bit IV). This allows for maximizing Draco's resource efficiency by easily adapting the hardware implementation to each device's specific capabilities. For example, if the secret key is burned into the device or stored in an EEPROM (a common RFID scenario [AHM14], assumed, e.g., by Plantlet) and the IV is constituted by the device's frame counter (as, e.g., in A5/1), then no storage cells for this data need to be allocated inside of the Draco hardware module, leading to the most lightweight variant labeled Draco in Table 2. If, on the other hand, the 32-bit key prefix and the 96-bit IV should both be available only at the beginning of state initialization (as generally assumed by Grain-128a), additional storage cells are required, leading to Draco$_{[KI]}$. The variants Draco$_{[K]}$ resp. Draco$_{[I]}$ represent the two intermediate scenarios that only the 32-bit key prefix resp. the 96-bit IV need to be held locally in the Draco hardware module.

The numbers presented in Table 2 show that the Draco stream cipher is likewise attractive for lightweight RFID and highspeed computation scenarios. For example, when making optimal use of an RFID tag's resources (i.e., burned/EEPROM key, transmission counter as IV), Draco requires 23 % less area (2142 vs. 2795 GE) and 31 % less power (79.2 vs. 115.3 µW) than Grain-128a at a clock frequency of 10 MHz. In the case of high speed computing, on the other hand, everything comes down to energy consumption. At a clock frequency of 1 GHz, all four implementation variants of Draco consume about 34 % less energy than Grain-128a for producing 10 kbit of keystream (including state initialization). In particular, this substantial advantage is achieved even if the 32-bit key

prefix and the 96-bit IV have to be stored locally inside of the Draco hardware module (i.e., 32.7 nJ for Draco$_{[KI]}$ vs. 50.4 nJ for Grain-128a; cf. Appendix C.1).

In direct comparison to Atom we can see that Draco needs 28 % less area (2142 vs. 2976 GE) and 24 % less power (79.2 vs 104.9 µW) at a clock frequency of 10 MHz. Further comparing Atom$_{[K]}$ to Draco$_{[KI]}$ we see improvements of 21 % in area (3025 vs 3858 GE) and an improvement of 20 % in power consumption (100.6 vs 126.1 µW) at a clock frequency of 10 MHz.

The reason behind this is that already for moderate clock frequencies (here: between 10 MHz and 20 MHz) the dynamic power consumption (due to switching of values) dominates the static power consumption (due to leakage) of flip-flop storage cells. To the best of our knowledge, this effect has never been considered in stream cipher design before. Instead, the classical design paradigm (e.g., followed by Grain-128a, but also by Plantlet and Lizard) exclusively focused on the *number* of flip-flops, ignoring their actual *usage*. With Draco$_{[KI]}$ we demonstrate that even if a $2n$-bit storage is required inside the cipher hardware module to achieve n-bit security against TMDTO attacks, algorithmically keeping half of this state constant is much more efficient (cf. Tab. 2) than and equally secure (see Section 8 and Section 5) as constantly updating the whole of it.

10 Conclusion

In this work we presented the new generic stream cipher construction CIVK and a new stream cipher proposal called Draco that instantiates CIVK. CIVK provably provides *full* volatile state length security against distinguishing attacks providing a solid theoretical foundation to design stream ciphers upon.

Draco uses a 128-bit key, which is loaded to the volatile state cells of its feedback shift registers during initialization. A 32-bit prefix of this key, together with a 96-bit initial value, is continuously employed as part of the state update during keystream generation. If the key prefix and the initial value are stored 'externally' (e.g., inside an EEPROM), this design requires 23 % less area and 31 % less power than Grain-128a at 10 MHz.

For high-performance environments, we also considered an implementation variant called Draco$_{[KI]}$ with the key prefix and the initial value stored inside the cipher hardware module, while still only half of the total internal state is updated during state updates. When clocked at 1 GHz, this variant consumes about 34 % less energy than Grain-128a, still providing 128 bits of security and thus challenging the current paradigm of stream ciphers to always incorporate all internal state bits during state updates.

As future work we suggest to evaluate the performance of Draco 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 Draco are possible, for example by choosing a lighter output function.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. The authors also thank Yann Rotella for shepherding the final version of this paper. Further, we thank Tobias Horn for his analysis of Draco's resistance against cube attacks.

References

[AGH18] Vahid Amin Ghafari and Honggang Hu. Fruit-80: A Secure Ultra-lightweight Stream Cipher for Constrained Environments. *Entropy*, 20(3):180, 2018.

[ÅHJM11] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: A New Version of Grain-128 with Optional Authentication. *IJWMC*, 5(1):48–59, December 2011.

[AHM14] Frederik Armknecht, Matthias Hamann, and Vasily Mikhalev. Lightweight Authentication Protocols on Ultra-Constrained RFIDs - Myths and Facts. In *RFIDSec 2014*, pages 1–18. Springer International Publishing, Cham, 2014.

[AM15] Frederik Armknecht and Vasily Mikhalev. On Lightweight Stream Ciphers with Shorter Internal States. In *FSE 2015*, pages 451–470. Springer, 2015.

[Bab95] Steve H. Babbage. Improved "exhaustive search" attacks on stream ciphers. In *European Convertion on Security and Detection 1995*, pages 161–166, May 1995.

[BB06] Elad Barkan and Eli Biham. Conditional Estimators: An Effective Attack on A5/1. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, pages 1–19. Springer, Berlin, Heidelberg, 2006.

[BBI19] Subhadeep Banik, Khashayar Barooti, and Takanori Isobe. Cryptanalysis of Plantlet. *IACR Transactions on Symmetric Cryptology*, 2019, Issue 3:103–120, 2019.

[BCI+21] Subhadeep Banik, Andrea Caforio, Takanori Isobe, Fukang Liu, Willi Meier, Kosei Sakamoto, and Santanu Sarkar. Atom: A Stream Cipher with Double Key Filter. *IACR Transactions on Symmetric Cryptology*, pages 5–36, 2021.

[BD06] Steve Babbage and Matthew Dodd. The stream cipher MICKEY 2.0. eSTREAM: the ECRYPT Stream Cipher Project, 2006. http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf.

[BGJ09] Côme Berbain, Henri Gilbert, and Antoine Joux. Algebraic and Correlation Attacks against Linearly Filtered Non Linear Feedback Shift Registers. In *SAC 2008*, pages 184–198. Springer, Berlin, Heidelberg, 2009.

[BGM06] Côme Berbain, Henri Gilbert, and Alexander Maximov. Cryptanalysis of Grain. In *FSE 2006*, pages 15–29. Springer, Berlin, Heidelberg, 2006.

[BGW99] Marc Briceno, Ian Goldberg, and David Wagner. A pedagogical implementation of A5/1, 1999. Available at http://www.scard.org/gsm/a51.html.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, pages 1–13. Springer, Berlin, Heidelberg, 2000.

[BSW01] Alex Biryukov, Adi Shamir, and David Wagner. Real Time Cryptanalysis of A5/1 on a PC. In *FSE 2000*, pages 1–18. Springer, Berlin, Heidelberg, 2001.

[CP05] Christophe De Cannière and Bart Preneel. Trivium – Specifications. eSTREAM: the ECRYPT Stream Cipher Project, 2005. http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf.

[CS14] Shan Chen and John Steinberger. Tight Security Bounds for Key-alternating Ciphers. In *EUROCRYPT 2014*, pages 327–350. Springer, 2014.

[DCKP08] Christophe De Cannière, Özgül Küçük, and Bart Preneel. Analysis of Grain's Initialization Algorithm. In Serge Vaudenay, editor, *AFRICACRYPT 2008*, pages 276–289. Springer, Berlin, Heidelberg, 2008.

[DH15] Elena Dubrova and Martin Hell. Espresso: A stream cipher for 5G wireless communication systems. *Cryptography and Communications*, pages 1–17, 2015.

[DR08] 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.

[DS11] Itai Dinur and Adi Shamir. Breaking Grain-128 with Dynamic Cube Attacks. In *FSE 2011*, pages 167–187. Springer, Berlin, Heidelberg, 2011.

[DV18] Alex De Vries. Bitcoin's growing energy problem. *Joule*, 2(5):801–805, 2018.

[ECR08] ECRYPT – European Network of Excellence for Cryptology. eSTREAM: the ECRYPT stream cipher project, 2008. http://www.ecrypt.eu.org/stream/.

[EK15] Muhammed F Esgin and Orhun Kara. Practical Cryptanalysis of Full Sprout with TMD Tradeoff Attacks. In *International Conference on Selected Areas in Cryptography*, pages 67–85. Springer, 2015.

[Fel07] 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.

[FMS01] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In Serge Vaudenay and Amr M. Youssef, editors, *SAC 2001*, pages 1–24. Springer, Berlin, Heidelberg, 2001.

[GB08] 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.

[GCB06] 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.

[GGK06] Berndt 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.

[Gol96] Jovan Dj. Golić. On the security of nonlinear filter generators. In Dieter Gollmann, editor, *FSE 1996*, pages 173–188. Springer, Berlin, Heidelberg, 1996.

[Hel80] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, Jul 1980.

[HJM06] 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.

[HJM+19] Martin Hell, Thomas Johansson, Willi Meier, Jonathan Sönnerup, and Hirotaka Yoshida. An AEAD Variant of the Grain Stream Cipher. In *C2SI*, pages 55–71, Cham, 2019. Springer International Publishing.

[HJMM08] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The Grain Family of Stream Ciphers. In *New Stream Cipher Designs: The eSTREAM Finalists*, pages 179–190. Springer, Berlin, Heidelberg, 2008.

[HK15] Matthias Hamann and Matthias Krause. On Stream Ciphers with Provable Beyond-the-Birthday-Bound Security against Time-Memory-Data Tradeoff Attacks. Cryptology ePrint Archive, Report 2015/636, 2015. http://eprint.iacr.org/2015/636.

[HKM17a] Matthias Hamann, Matthias Krause, and Willi Meier. A Note on Stream Ciphers that Continuously Use the IV. *IACR Cryptology ePrint Archive*, 2017:1172, 2017.

[HKM17b] Matthias Hamann, Matthias Krause, and Willi Meier. LIZARD – A Lightweight Stream Cipher for Power-constrained Devices. *IACR ToSC*, 2017(1):45–79, 2017.

[HKM19] Matthias Hamann, Matthias Krause, and Alexander Moch. Tight Security Bounds for Generic Stream Cipher Constructions. In *SAC 2019*, pages 335–364. Springer, 2019.

[HKMZ18] Matthias Hamann, Matthias Krause, Willi Meier, and Bin Zhang. Design and Analysis of Small-state Grain-like Stream Ciphers. *Cryptography and Communications*, 10(5):803–834, 2018.

[Hor21] Tobias Horn. On Cube Attacks on Stream Ciphers. Master's thesis, Universität Mannheim, 2021. https://www.wim.uni-mannheim.de/media/Lehrstuehle/wim/ths/files/tohorn_masters.pdf.

[HS05] Jin Hong and Palash Sarkar. New applications of time memory data tradeoffs. In Bimal Roy, editor, *ASIACRYPT 2005*, pages 353–372, Berlin, Heidelberg, 2005. Springer.

[Ins97] Institute of Electrical and Electronics Engineers. 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.

[Ins04] Institute of Electrical and Electronics Engineers. 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.

[Ins21] Institute of Electrical and Electronics Engineers. 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 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pages 1–4379, 2021.

[Kra02] Matthias Krause. BDD-Based Cryptanalysis of Keystream Generators. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, pages 222–237. Springer, Berlin, Heidelberg, 2002.

[KSV13] D. Karaklajić, J. Schmidt, and I. Verbauwhede. Hardware designer's guide to fault attacks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(12):2295–2306, 2013.

[Küç06] 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.

[LM12] Michael Lehmann and Willi Meier. Conditional Differential Cryptanalysis of Grain-128a. In *CANS 2012*, pages 1–11. Springer, Berlin, Heidelberg, 2012.

[LMV05] Yi Lu, Willi Meier, and Serge Vaudenay. The Conditional Correlation Attack: A Practical Attack on Bluetooth Encryption. In Victor Shoup, editor, *CRYPTO 2005*, pages 97–117. Springer, Berlin, Heidelberg, 2005.

[MAM16] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On Ciphers that Continuously Access the Non-volatile Key. *IACR ToSC*, pages 52–79, 2016.

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts. In *EUROCRYPT 2016*, pages 311–343. Springer, Berlin, Heidelberg, 2016.

[MS89] Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989.

[Pat08] Jacques Patarin. The "coefficients H" technique. In *SAC 2008*, pages 328–345. Springer, 2008.

[Pop15] A. Popov. Prohibiting RC4 Cipher Suites. RFC 7465 (Proposed Standard), February 2015.

[Res18] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[Sch95] Bruce Schneier. *Applied Cryptography (2nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[Sie85] Thomas Siegenthaler. Decrypting a Class of Stream Ciphers Using Ciphertext Only. *IEEE Transactions on Computers*, 34(1):81–85, January 1985.

[SIG14] Bluetooth SIG. Bluetooth Core Specification 4.2, 2014. https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=286439.

[Ste07] Dirk Stegemann. Extended BDD-Based Cryptanalysis of Keystream Generators. In *SAC 2007*, pages 17–35. Springer, Berlin, Heidelberg, 2007.

[TIHM17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube Attacks on Non-Blackbox Polynomials Based on Division Property. In *CRYPTO 2017*, pages 250–279, Cham, 2017. Springer International Publishing.

[TIM+18] Yosuke Todo, Takanori Isobe, Willi Meier, Kazumaro Aoki, and Bin Zhang. Fast Correlation Attack Revisited. In *CRYPTO 2018*, pages 129–159, Cham, 2018. Springer International Publishing.

[TMA20] Yosuke Todo, Willi Meier, and Kazumaro Aoki. On the Data Limitation of Small-State Stream Ciphers: Correlation Attacks on Fruit-80 and Plantlet. In *SAC 2019*, pages 365–392, Cham, 2020. Springer International Publishing.

[WLLM19] Shichang Wang, Meicheng Liu, Dongdai Lin, and Li Ma. Fast Correlation Attacks on Grain-like Small State Stream Ciphers and Cryptanalysis of Plantlet, Fruit-v2 and Fruit-80. Cryptology ePrint Archive, Report 2019/763, 2019. https://eprint.iacr.org/2019/763.

[ZGM17] Bin Zhang, Xinxin Gong, and Willi Meier. Fast Correlation Attacks on Grain-like Small State Stream Ciphers. *IACR ToSC*, 2017(4):58–81, Dec. 2017.

[ZW09] Haina Zhang and Xiaoyun Wang. Cryptanalysis of Stream Cipher Grain Family. Cryptology ePrint Archive, Report 2009/109, 2009. http://eprint.iacr.org/2009/109.

A Test Vectors

Key (128 bits), IV (96 bits) and the corresponding first 128 keystream bits in hexadecimal notation. To avoid ambiguity, note that, e.g., the key

`0x01234FFFFFFFFFFFFFFFFFFFFFFFFFFF`

corresponds to

$$(K_0, \ldots, K_{127}) = (0,0,0,0,0,0,0,1,0,0,1,0,0,0,1,1,0,1,0,0,1,\ldots,1)\,.$$

Similarly, for the keystream, the example

`0x01000000000000000000000000000000`

would mean that the first seven keystream bits (i.e., $z_{512}, \ldots, z_{518}$) are zero, followed by a one bit and 120 more zero bits.

```
Key:       0x00000000000000000000000000000000
IV:        0x000000000000000000000000
Keystream: 0x6FB3AB21A9B00507CE18710E35FB40AB

Key:       0x0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F
IV:        0xF0F0F0F0F0F0F0F0F0F0F0F0
Keystream: 0xD065AC7B058A2B56523BAC08DE9E93A4

Key:       0x0123456789ABCDEF9876543210FEDCBA
IV:        0xAABCDEF0123456789ABCDEFF
Keystream: 0x45A84DC6F56623EF482989B15E924ED8
```

B Output Function a: Tap Selection

As pointed out in Section 7.4, 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 [Gol96] to assess the security of nonlinear filter generators consisting of a single LFSR and a nonlinear output function. A similar approach was taken, e.g., for the NFSR-based stream cipher Espresso [DH15] and for the Grain-like small-state stream cipher Lizard [HKM17b].

Golić defines "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)" [Gol96] and, as a security criterion for output functions, requires that the taps "should be chosen according to a full or a λth-order positive difference set, with λ as small as possible" [Gol96].

In line with this, the output function a of Draco has the following properties:

- No taps from NFSR1, except some of those in the additionally required term of f_1, $\oplus\neg S_1^t\neg S_2^t\cdots\neg S_{31}^t\neg S_{32}^t$ (cf. Sec. 6.1), are used at the same time for its feedback function f_1 and the output function. (In Grain-128a, the feedback function of the LFSR, which corresponds to NFSR1 in our construction, and the output function do not share any taps, either.)

- The set
$$\{5, 11, 19, 22, 26, 31\}$$
of the tap indices (all from NFSR1) of $\mathcal{T}_t^{(2)}$ is a full positive difference set. This means that each two bits of the internal bitstream of NFSR1 never appear more than once together as part of this triangular function.

- No taps from NFSR2 are used at the same time for its feedback function and the output function. (In Grain-128a, the feedback function of the NFSR, which corresponds to NFSR2 in our construction, and the output function share only a single tap called "b_{i+95}" in [ÅHJM11].)

- The direct sum $\mathcal{L}_t + \mathcal{Q}_t + \mathcal{T}_t^{(1)}$ 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. Section 8.2.)

- The set
$$\{7, 15, 32, 47, 53, 66, 76, 80, 92\}$$
of the tap indices (all from NFSR2) of the linear monomials of $\mathcal{L}_t + \mathcal{T}_t^{(1)} + \mathcal{T}_t^{(3)}$ is a full positive difference set.

- The set
$$\{5, 12, 20, 34, 38, 44, 57, 69, 74, 85\}$$
of the tap indices (all from NFSR2) of the quadratic monomials of $\mathcal{Q}_t + \mathcal{T}_t^{(1)}$ 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
$$\{|5-85|, |12-74|, |20-69|, |34-57|, |38-44|\}$$
of differences between the two taps (all from NFSR2) of each quadratic monomial in $\mathcal{Q}_t + \mathcal{T}_t^{(1)}$ and
$$\{|3-67|, |11-13|, |17-18|, |27-59|, |36-39|, |40-48|,\\ |50-79|, |54-71|, |58-63|, |61-65|, |68-84|\}$$
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
$$\{|5-85|, |12-74|, |20-69|, |34-57|, |38-44|\}$$
between the two taps (all from NFSR2) of each quadratic monomial in $\mathcal{Q}_t + \mathcal{T}_t^{(1)}$ appears as a difference between two taps of a higher degree monomial of $\mathcal{T}_t^{(1)}$.

- Each of the sets

$$\{23, 49, 83\},$$
$$\{6, 33, 51, 73\},$$
$$\{4, 29, 43, 60, 81\},$$
$$\{9, 14, 35, 42, 55, 77\},$$
$$\{1, 16, 28, 45, 64, 75, 88\}$$

  of tap indices (all from NFSR2) of the monomials of degree $3, \ldots, 7$ of $\mathcal{T}_t^{(1)}$ is a full positive difference set. Consequently, each two bits of the internal bitstream of NFSR2 never appear more than once together as part of each (i.e., the same) of those monomials.

C Implementation Details

C.1 Hardware Evaluation Setup

As done by Hell et al. for their hardware evaluation of Grain-128AEAD [HJM+19] (a current round-2 candidate in the NIST Lightweight Cryptography Standardization process), we target 0.65 nm CMOS process technology and use Synopsys tools for synthesis and power estimation. More precisely, our results (see Table 2 in Section 9) are obtained via Synopsys Design Compiler 2018.06-SP4 and are based on the netlist generated for the Draco reference implementation (see Appendix C.2) employing TSMC's `tcbn65gplus 200a` standard cell library.

Like Feldhofer in [Fel07] for his low-power implementations of Trivium and Grain, we employ clock gating, which is a standard technique for reducing dynamic power consumption in synchronous circuits. In a nutshell, this means that while an edge-triggered flip-flop is not supposed to switch values (such as the registers holding the 32-bit key prefix and the 96-bit IV in $\text{Draco}_{[\text{KI}]}$ for $t \geq 1$), its enable port is disconnected from the circuit's clock signal.

The switching activity for power estimation (recorded with Synopsys VCS 2018.09-SP1-1 and fed back to Design Compiler) covers the generation of 10 kbit of keystream (as done by Good and Benaissa in [GCB06] in their hardware comparison of eSTREAM candidates) and includes the state initialization of the compared cipher modules. To improve the accuracy of the results, switching activity for 100 different random key/IV combinations is considered and the arithmetic mean of the respective power estimates is computed.

As, after state initialization, all cipher implementations compared in Section 9 produce one keystream bit per clock cycle, energy consumption can be straightforwardly computed and compared on the basis of power estimates. The only thing which has to be taken into account here is that Grain-128a performs 256 initialization rounds as compared to 512 rounds for Draco. Consequently, the amount of energy required for producing, 10 kbit of keystream (including state initialization) at a clock speed of 1 GHz is computed as $(10256/(1\text{ GHz})) \cdot (4912.9\ \mu\text{W}) = 50.4$ nJ for Grain-128a and as $(10512/(1\text{ GHz})) \cdot (3119.3\ \mu\text{W}) = 32.7$ nJ for $\text{Draco}_{[\text{KI}]}$.

The critical path delay of a circuit determines the maximum possible clock frequency and, hence, the maximum achievable throughput. The worst delay for any of our four implementation variants of Draco is 560 ps, which corresponds to an achievable clock frequency of about 1.8 GHz. Also note that where encryption throughputs even larger than 1.8 Gbit/s are required, the delay can be further reduced by using techniques like pipelining (as done for the stream cipher Espresso in [DH15]). 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.

Finally, note that in terms of chip area, the three implementation variants Draco, Draco[K] and Draco[I] (see Section 9) would additionally benefit from an external key and/or IV source (like an EEPROM) which takes over the task of key/IV bit selection based on an index or supplies the key/IV bits sequentially. However, for reasons of fairness, in our hardware evaluation we assumed the (from Draco's point of view *worst*) situation that all key and IV bits are provided via separate wires to the cipher module, which then has to take care of key/IV bit selection itself for computing the key-IV-schedule bit (cf. reference implementation in Appendix C.2).

C.2 Reference Implementation

Listing 1: Reference implementation of Draco in Verilog.

```
1  `timescale 1us / 1ps
2
3  //************************************
4
5  `define KEY_EXTERNAL
6  `define IV_EXTERNAL
7
8  //************************************
9
10 module draco(
11   input wire clk,
12   input wire reset,
13   input wire enable,
14   input wire [0:127] key,
15   input wire [0:95] iv,
16   output wire keystreamBit,
17   output wire keystreamFlag
18   );
19
20 ////////////////
21
22 wire [0:31] keyPrefix_local;
23 `ifdef KEY_EXTERNAL
24   assign keyPrefix_local = key[0:31];
25 `else
26   reg [0:31] keyPrefix_local_reg;
27   assign keyPrefix_local = keyPrefix_local_reg;
28 `endif
29
30 wire [0:96] iv_local;
31 assign iv_local[0] = 1'b0;
32 `ifdef IV_EXTERNAL
33   assign iv_local[1:96] = iv;
34 `else
35   reg [0:95] iv_local_reg;
36   assign iv_local[1:96] = iv_local_reg;
37 `endif
38
39 ////////////////
40
41 reg [0:0] cipherFSM;
42
43 localparam S_PHASE2 = 1'b0;
44 localparam S_GENOUT = 1'b1;
45
46 ////////////////
47
48 assign keystreamFlag = cipherFSM[0];
49
50 ////////////////
```

```
51
52 reg [0:32] nfsr1_state;
53 reg [0:94] nfsr2_state;
54
55 ////////////////
56
57 wire nfsr1_feedbackBit;
58
59 // efficient implementation of NFSR1's feedback function as decribed in the
       ACHTERBAHN paper (cf. design specs)
60
61 wire nfsr1_feedbackBit_achterbahn;
62 wire nfsr1_maj, nfsr1_mux1, nfsr1_mux2, nfsr1_mux3, nfsr1_mux4,
       nfsr1_avoidAllZeroTerm;
63
64 assign nfsr1_maj = (nfsr1_state[1] & nfsr1_state[14]) | (nfsr1_state[1] &
       nfsr1_state[18]) | (nfsr1_state[14] & nfsr1_state[18]);
65 assign nfsr1_mux1 = (nfsr1_state[13]) ? nfsr1_state[15] : nfsr1_state[25];
66 assign nfsr1_mux2 = (nfsr1_state[16]) ? nfsr1_state[12] : nfsr1_state[15];
67 assign nfsr1_mux3 = (nfsr1_state[17]) ? nfsr1_state[24] : nfsr1_state[8];
68 assign nfsr1_mux4 = (nfsr1_mux3) ? nfsr1_maj : nfsr1_mux2;
69
70 // NOR reduction of nfsr1_state[1:32]
71 assign nfsr1_avoidAllZeroTerm = ~|nfsr1_state[1:32];
72
73 assign nfsr1_feedbackBit_achterbahn = nfsr1_state[0] ^ nfsr1_state[2] ^
       nfsr1_state[7] ^ nfsr1_state[9] ^ nfsr1_state[10] ^ nfsr1_state[23] ^
       nfsr1_state[30] ^ (nfsr1_state[15] & nfsr1_state[16]) ^ nfsr1_mux1 ^
       nfsr1_mux4;
74
75 assign nfsr1_feedbackBit = nfsr1_feedbackBit_achterbahn ^
       nfsr1_avoidAllZeroTerm;
76
77 ////////////////
78
79 wire nfsr2_feedbackBit;
80
81 assign nfsr2_feedbackBit = nfsr2_state[0] ^ nfsr2_state[26] ^
       nfsr2_state[56] ^ nfsr2_state[89] ^ nfsr2_state[94] ^ (nfsr2_state[3] &
       nfsr2_state[67]) ^ (nfsr2_state[11] & nfsr2_state[13]) ^
       (nfsr2_state[17] & nfsr2_state[18]) ^ (nfsr2_state[27] &
       nfsr2_state[59]) ^ (nfsr2_state[36] & nfsr2_state[39]) ^
       (nfsr2_state[40] & nfsr2_state[48]) ^ (nfsr2_state[50] &
       nfsr2_state[79]) ^ (nfsr2_state[54] & nfsr2_state[71]) ^
       (nfsr2_state[58] & nfsr2_state[63]) ^ (nfsr2_state[61] &
       nfsr2_state[65]) ^ (nfsr2_state[68] & nfsr2_state[84]) ^
       (nfsr2_state[8] & nfsr2_state[46] & nfsr2_state[87]) ^ (nfsr2_state[22]
       & nfsr2_state[24] & nfsr2_state[25]) ^ (nfsr2_state[70] &
       nfsr2_state[78] & nfsr2_state[82]) ^ (nfsr2_state[86] & nfsr2_state[90]
       & nfsr2_state[91] & nfsr2_state[93]);
82
83 ////////////////
84
85 wire outLin, outQuad, outTri1, outTri2, outTri3;
86
87 assign outLin = nfsr2_state[7] ^ nfsr2_state[15] ^ nfsr2_state[32] ^
       nfsr2_state[47] ^ nfsr2_state[66] ^ nfsr2_state[80] ^ nfsr2_state[92];
88
89 assign outQuad = (nfsr2_state[5] & nfsr2_state[85]) ^ (nfsr2_state[12] &
       nfsr2_state[74])^ (nfsr2_state[20] & nfsr2_state[69]) ^
       (nfsr2_state[34] & nfsr2_state[57]);
90
91 assign outTri1 = nfsr2_state[53]  ^ (nfsr2_state[38] & nfsr2_state[44]) ^
       (nfsr2_state[23] & nfsr2_state[49] & nfsr2_state[83]) ^ (nfsr2_state[6]
       & nfsr2_state[33] & nfsr2_state[51] & nfsr2_state[73]) ^
       (nfsr2_state[4] & nfsr2_state[29] & nfsr2_state[43] & nfsr2_state[60] &
```

```
    nfsr2_state[81]) ^ (nfsr2_state[9] & nfsr2_state[14] & nfsr2_state[35]
    & nfsr2_state[42] & nfsr2_state[55] & nfsr2_state[77]) ^
    (nfsr2_state[1] & nfsr2_state[16] & nfsr2_state[28] & nfsr2_state[45] &
    nfsr2_state[64] & nfsr2_state[75] & nfsr2_state[88]);
92
93  assign outTri2 = nfsr1_state[26] ^ (nfsr1_state[5] & nfsr1_state[19]) ^
    (nfsr1_state[11] & nfsr1_state[22] & nfsr1_state[31]);
94
95  assign outTri3 = nfsr2_state[76] ^ (nfsr1_state[3] & nfsr2_state[10]) ^
    (nfsr1_state[20] & nfsr2_state[21] & nfsr2_state[30]) ^ (nfsr1_state[6]
    & nfsr1_state[29] & nfsr2_state[62] & nfsr2_state[72]);
96
97  assign keystreamBit = outLin ^ outQuad ^ outTri1 ^ outTri2 ^ outTri3;
98
99  ////////////////
100
101 reg [8:0] ctr;
102
103 wire [4:0] keyPrefixIndex;
104 assign keyPrefixIndex = ctr[4:0];
105
106 reg [6:0] ivCtr;
107
108 wire [6:0] ivIndex;
109 assign ivIndex = ivCtr;
110
111 wire done_mixing;
112 assign done_mixing = (ctr == 9'd511) ? 1'b1 : 1'b0;
113
114 ////////////////
115
116 wire kisBit;
117 assign kisBit = keyPrefix_local[keyPrefixIndex] ^ iv_local[ivIndex];
118
119 // KIS bit computation different during first and second half of mixing
120 wire kisBitMixing;
121 assign kisBitMixing = (ctr[8] == 1'b1) ? kisBit : iv_local[ivIndex];
122
123 ////////////////
124
125 always @(posedge clk)
126   begin
127     if (reset)
128       begin
129         `ifndef KEY_EXTERNAL
130           keyPrefix_local_reg <= key[0:31];
131         `endif
132
133         `ifndef IV_EXTERNAL
134           iv_local_reg <= iv;
135         `endif
136
137         nfsr2_state[0] <= ~key[0];
138         nfsr2_state[1:94] <= key[1:94];
139         nfsr1_state[0:32] <= key[95:127];
140
141         ctr <= 9'd0;
142         ivCtr <= 7'd0;
143
144         cipherFSM <= S_PHASE2;
145       end
146     else if (enable)
147       begin
148         nfsr1_state[0:31] <= nfsr1_state[1:32];
149
150         nfsr2_state[0:93] <= nfsr2_state[1:94];
```

```
151
152         // if ivCtr == 96 --> set ivCtr := 0
153         if (ivCtr[6] && ivCtr[5])
154           ivCtr[6:5] <= 2'd0;
155         else
156           ivCtr <= ivCtr + 1;
157
158         case (cipherFSM)
159           S_PHASE2:
160             begin
161               nfsr1_state[32] <= nfsr1_feedbackBit ^ keystreamBit;
162               nfsr2_state[94] <= nfsr2_feedbackBit ^ nfsr1_state[0] ^
                      kisBitMixing ^ keystreamBit;
163
164               if (done_mixing)
165                 begin
166                   ctr <= 9'd0;
167                   cipherFSM <= S_GENOUT;
168                 end
169               else
170                 ctr <= ctr + 1;
171             end
172
173           S_GENOUT:
174             begin
175               nfsr1_state[32] <= nfsr1_feedbackBit;
176               nfsr2_state[94] <= nfsr2_feedbackBit ^ nfsr1_state[0] ^
                      kisBit;
177
178               ctr[6:0] <= ctr[6:0] + 1;
179             end
180         endcase
181       end
182   end
183
184 ////////////////
185
186 endmodule
187
188 //*************************************
```