IACR Transactions
ISSN 2519-173X, Vol. 0, No. 0, pp. 0–0. DOI:10.13154/tosc.v0.i0.0-0

Atom: A Stream Cipher with Double Key Filter

Subhadeep Banik[1], Andrea Caforio[1], Takanori Isobe[2,3,4], Fukang Liu[2,5], Willi Meier[6], Kosei Sakamoto[2], Santanu Sarkar[7]

[1] LASEC, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland. {subhadeep.banik,andrea.caforio}@epfl.ch
[2] University of Hyogo, Kobe, Japan. takanori.isobe@ai.u-hyogo.ac.jp, liufukangs@gmail.com, k.sakamoto0728@gmail.com
[3] NICT, Tokyo, Japan
[4] PRESTO, Japan Science and Technology Agency, Tokyo, Japan
[5] East China Normal University, Shanghai, China
[6] FHNW, Windisch, Switzerland. willimeier48@gmail.com
[7] Indian Institute of Technology, Madras, India. santanu@iitm.ac.in

Abstract. It has been common knowledge that for a stream cipher to be secure against generic TMD tradeoff attacks, the size of its internal state in bits needs to be at least twice the size of the length of its secret key. In FSE 2015, Armknecht and Mikhalev however proposed the stream cipher Sprout with a Grain-like architecture, whose internal state was equal in size with its secret key and yet resistant against TMD attacks. Although Sprout had other weaknesses, it germinated a sequence of stream cipher designs like Lizard and Plantlet with short internal states. Both these designs have had cryptanalytic results reported against them. In this paper, we propose the stream cipher Atom that has an internal state of 159 bits and offers a security of 128 bits. Atom uses two key filters simultaneously to thwart certain cryptanalytic attacks that have been recently reported against keystream generators. In addition, we found that our design is one of the smallest stream ciphers that offers this security level, and we prove in this paper that Atom resists all the attacks that have been proposed against stream ciphers so far in literature. On the face of it, Atom also builds on the basic structure of the Grain family of stream ciphers. However, we try to prove that by including the additional key filter in the architecture of Atom we can make it immune to all cryptanalytic advances proposed against stream ciphers in recent cryptographic literature.

Keywords: Short State Stream Cipher, Lightweight Cryptography, Grain family

1 Introduction

The technique of using Time Memory Tradeoffs to invert one-way functions was introduced by Hellman in his seminal paper [Hel80] on the topic in 1980. This was a chosen plaintext attack and attempted to invert the one-way function which maps the keyspace to the ciphertext space by encrypting a chosen message using a block-cipher. The work of Babbage [Bab95], Golic [Gol97] and Biryukov-Shamir [BS00] applied such tradeoffs to the one-way functions that mapped the internal state space to a keystream segment of a stream cipher. In [BS00], it was proven that for a stream cipher to be secure against generic Time-Memory-Data (TMD) tradeoff attacks, the size of its internal state needed to be at least twice the size of its secret key in bits. The authors mounted a table based attack that stored functional iterates of a function that mapped the internal state of the cipher to a keystream prefix of equal length. Since then there have been several variants of this attack: in [HS05] the authors used the function that mapped the string consisting of

Key and IV to the corresponding length keystream prefix to attack the A5/3 stream cipher and [DK08] which used the function that maps the key to the corresponding keystream prefix of same length for a fixed set of IVs. In FSE 2015, the stream cipher Sprout [AM15] was proposed that had a Grain-like architecture [HJM07] and surprisingly an internal state and key both of 80 bits. Yet the cipher seemed to be immune to any of the above TMD attacks. The reason for that is that although the internal state of Sprout was of 80 bits, the state update algorithm required a key input. And so it was impossible to construct any function that mapped internal state to keystream without the secret key.

Sprout was however cryptanalyzed in the following papers [LN15, Ban15, EK15, ZGM17]. The most important attack from a design point of view was reported in [EK15]. The function of the key that was used to update the state in Sprout was non-linear and so the authors were able to construct a table using some special states for which the key input was 0 for 40 consecutive cycles: hence a function mapping state to a 40-bit keystream prefix was possible for these special states. Using this the authors constructed a probabilistic attack that sampled random keystream prefix until they found one that was present in the tables. After this key recovery was possible in practical time. The attack underscored the fact that any key function that was used for state update had to be linear. This was exactly the approach used in later Grain-like designs like Lizard [HKM17] and Plantlet [MAM16].

Over the years there have been attacks reported against Lizard and Plantlet too. In [BICG17], the authors reported various distinguishing attacks against the full round version cipher and a key recovery attack against 223 of the 256 initialization rounds of the cipher. In [MSS$^+$18], a TMD tradeoff attack was proposed that found the internal state of the cipher. Very recently [BBI19] proposed a differential attack against Plantlet that finds the key in around 2^{70} encryptions. In [TMA19] a correlation attack was proposed on Plantlet, albeit on a version that allowed more keystream to be extracted from a single key-IV pair than mentioned in the specifications. There even have been attacks reported against the Grain family: [TIM$^+$18] reported a correlation attack against all 3 variants of the Grain family (Grain v1, Grain 128, Grain 128a), while [ZXM18] reported a near collision attack on Grain v1 based on recurrence of internal states with small internal difference.

Almost all stream ciphers with the Grain structure (including Grain v1 and Grain 128) have had some weaknesses or undesirable properties reported against them. Thus from a design point of view it is an important question whether it is possible to design stream ciphers securely in the long run. In this paper we take up this very challenge. Our primary goal was to see if we could improvise a circuit component that is able to prevent some of the attacks that have been proposed in literature. A secondary goal was to minimize the hardware footprint of the cipher, and keep it ideally less than other stream ciphers proposed in literature that offer a similar security level. We propose the stream cipher Atom that offers 128 bit security and has an internal state of around 159 bits (which is less than 25% more than the secret key size). On the face of it, Atom has the same Grain like structure adopted by Plantlet, Sprout and Lizard. However we do add a circuit-level novelty to the cipher specification that costs only around 150 Gate-Equivalents (GE) in hardware but guarantees immunity against all attacks proposed against small state ciphers in the recent past. In fact, we prove its security in the context of specific attacks that have been proposed against similar designs in literature and present implementation results that establish its competitiveness among ciphers popular in the symmetric cryptology community.

Note that what the additional filter essentially does, is update the internal state of the generator with a part of the secret key that depends on the internal state itself. One utilitarian aspect of this feature is that it prevents correlation-like attacks by misaligning the key bits in every window. Some similar ideas of using such state-dependent update in some sense has been previously seen in A5/1 [Can11] and RC4 [Fon11]. In A5/1, the update routine first calculates the majority function over three specific control bits from the three LFSRs in its internal state. Each register is clocked if and only if the control bit of the register equals the majority bit. Also it is well known that RC4 selects elements to swap in its internal state using the index j which is computed as $(j + S[i]) \bmod 256$. Although the idea is similar these state dependent updates do not provide immunity against correlation attacks in the same way that a state dependent key filter does.

1.1 Contributions and Organization of the Paper

The main contribution of the paper is small state stream cipher Atom. Although it has the same grain like structure seen in many past designs, the novelty of Atom is an additional key filter that is used to update the NFSR update function. The bits controlling the additional filter is taken from the LFSR that not only increases the period of the keystream but solves many genres of cryptanalytic advances reported against short state stream ciphers in the past. In particular Atom has the following features:

- Security level of 128 bits.
- Immune against generic TMD tradeoff attacks, and also those against Sprout [EK15].
- Immune against correlation attacks like those against Plantlet [TMA19].
- Immune against key recovery attacks like those against Plantlet [BBI19].
- Circuit area is competitive among hardware stream ciphers that provide the same security level.

In Section 2, we present the algebraic specifications of the cipher. In Section 3, we argue about some design choices and why they make sense for a short state cipher. Section 4, presents a comprehensive security analysis. Section 5 presents implementation results. Section 6 concludes the paper.

2 Specification

Atom is a stream cipher similar to the Grain family, which is composed of a linear feedback shift register (LFSR) and a nonlinear feedback shift register (NFSR). An overview of Atom can be referred to Figure 1. The size of the secret key k of Atom is 128 bits. Similar to most stream ciphers, there will be an initialization phase and a keystream generation phase in Atom.

2.1 Building Blocks

We begin the paper with a description of the building blocks of our construction. Atom has a grain like structure with an LFSR and NFSR connected through a xor gate as shown in Fig. 1. The length of NFSR and LFSR are 90 bits and 69 bits, respectively. At clock $t = 0, 1, \ldots$, denote the contents in NFSR and LFSR by $B^t = (b_0^t, \ldots, b_{89}^t)$ and $L^t = (l_0^t, \ldots, l_{68}^t)$, respectively. The definitions of the output function, LFSR and NFSR are specified in the following.

Output function. The output function $O(B^t, L^t)$ of Atom is a sum of linear terms, a quadratic bent function and another 9-variable function h. It is defined in the following way:

$$O(B^t, L^t) = \quad b_1^t \oplus b_5^t \oplus b_{11}^t \oplus b_{22}^t \oplus b_{36}^t \oplus b_{53}^t \oplus b_{72}^t \oplus b_{80}^t \oplus b_{84}^t$$

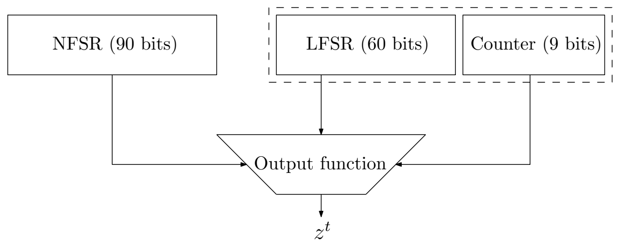

Figure 1: Overview of the construction. Note that during initialization the 69-bit LFSR is partitioned into 60 and 9 bits with each part updated by its own update logic as shown in this figure. During the keystream generation phase, both these parts function as a single 69-bit LFSR.

$$\oplus l_5^t l_{16}^t \oplus l_{13}^t l_{15}^t \oplus l_{30}^t l_{42}^t \oplus l_{22}^t l_{67}^t \oplus h(l_7^t, l_{33}^t, l_{38}^t, l_{50}^t, l_{59}^t, l_{62}^t, b_{85}^t, b_{41}^t, b_9^t),$$

where $h(X) = h(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ is defined as below.

$$\begin{aligned} h(X) = \quad & x_0x_1x_2x_7x_8 \oplus x_0x_1x_2x_7 \oplus x_0x_1x_2x_8 \oplus x_0x_1x_2 \oplus x_0x_1x_3x_7x_8 \\ & \oplus x_0x_1x_3x_7 \oplus x_0x_1x_4x_7x_8 \oplus x_0x_1x_4x_7 \oplus x_0x_1x_4x_8 \oplus x_0x_1x_4 \\ & \oplus x_0x_1x_5x_7x_8 \oplus x_0x_1x_5x_7 \oplus x_0x_1x_6x_7x_8 \oplus x_0x_1x_6x_8 \\ & \oplus x_0x_1x_7x_8 \oplus x_0x_1x_8 \oplus x_0x_2x_3x_7x_8 \oplus x_0x_2x_3x_7 \oplus x_0x_2x_3x_8 \\ & \oplus x_0x_2x_3 \oplus x_0x_2x_4x_7x_8 \oplus x_0x_2x_4x_8 \oplus x_0x_2x_5x_7x_8 \oplus x_0x_2x_5x_7 \\ & \oplus x_0x_2x_5x_8 \oplus x_0x_2x_5 \oplus x_0x_2x_6x_7x_8 \oplus x_0x_2x_6x_8 \oplus x_0x_2x_7x_8 \\ & \oplus x_0x_2x_8 \oplus x_0x_3x_4x_7x_8 \oplus x_0x_3x_4x_7 \oplus x_0x_3x_5x_7x_8 \oplus x_0x_3x_5x_7 \\ & \oplus x_0x_3x_6x_7x_8 \oplus x_0x_3x_6x_7 \oplus x_0x_3x_8 \oplus x_0x_3 \oplus x_0x_4x_5x_7x_8 \\ & \oplus x_0x_4x_5x_7 \oplus x_0x_4x_6x_7x_8 \oplus x_0x_4x_6x_8 \oplus x_0x_4x_7 \oplus x_0x_4 \\ & \oplus x_0x_5x_6x_7x_8 \oplus x_0x_5x_6x_7 \oplus x_0x_5x_7x_8 \oplus x_0x_5x_7 \oplus x_0x_6x_7 \\ & \oplus x_0x_6x_8 \oplus x_0x_7x_8 \oplus x_1x_2x_3x_7x_8 \oplus x_1x_2x_4x_7x_8 \oplus x_1x_2x_4x_8 \\ & \oplus x_1x_2x_5x_7x_8 \oplus x_1x_2x_6x_7x_8 \oplus x_1x_2x_6x_8 \oplus x_1x_2x_7 \oplus x_1x_2x_8 \\ & \oplus x_1x_2 \oplus x_1x_3x_4x_7x_8 \oplus x_1x_3x_5x_7x_8 \oplus x_1x_3x_6x_7x_8 \oplus x_1x_3x_7 \\ & \oplus x_1x_4x_5x_7x_8 \oplus x_1x_4x_5x_8 \oplus x_1x_4x_6x_7x_8 \oplus x_1x_4x_7 \oplus x_1x_4 \\ & \oplus x_1x_5x_6x_7x_8 \oplus x_1x_5x_6x_7 \oplus x_1x_5x_7x_8 \oplus x_1x_5x_7 \oplus x_1x_5x_8 \\ & \oplus x_1x_6x_7 \oplus x_1x_8 \oplus x_1 \oplus x_2x_3x_4x_7x_8 \oplus x_2x_3x_5x_7x_8 \oplus x_2x_3x_6x_7x_8 \\ & \oplus x_2x_4x_5x_7x_8 \oplus x_2x_4x_5x_8 \oplus x_2x_4x_6x_7x_8 \oplus x_2x_4x_7x_8 \oplus x_2x_4x_8 \\ & \oplus x_2x_5x_6x_7x_8 \oplus x_2x_5x_6x_8 \oplus x_2x_5x_8 \oplus x_2x_6x_7x_8 \oplus x_2x_6x_8 \\ & \oplus x_2x_7x_8 \oplus x_2 \oplus x_3x_4x_5x_7x_8 \oplus x_3x_4x_5x_7 \oplus x_3x_4x_6x_7x_8 \\ & \oplus x_3x_4x_6x_7 \oplus x_3x_5x_6x_7x_8 \oplus x_3x_5x_7x_8 \oplus x_3x_6x_7x_8 \oplus x_3x_6x_7 \\ & \oplus x_3x_7 \oplus x_3 \oplus x_4x_5x_6x_7x_8 \oplus x_4x_5x_6x_8 \oplus x_4x_6x_7x_8 \oplus x_4x_6x_8 \\ & \oplus x_4x_7 \oplus x_5x_7x_8 \oplus x_5 \oplus x_6 \oplus x_7x_8 \oplus x_7 \oplus x_8 \oplus 1 \end{aligned}$$

Note that h is a $(9, 5, 3, 240)$ function, i.e. it is of 9 variables, algebraic degree 5, its correlation immunity is 3 and its non-linearity is 240. It has one of the highest non-linearities among all 9 variable Boolean functions (as shown in [KY10] the highest known non-linearity among 9 variable Boolean functions is 242). Thus the output is a sum of linear terms, a quadratic bent function and h. Since bent functions are known to have highest non-linearity for even variable functions, this provides us adequate protection from correlation attacks (see Section 3.3).

NFSR. The definition of the update function $G(B^t)$ used in NFSR is specified as follows.

$$\begin{aligned} G(B^t) = \quad & b_0^t \oplus b_{24}^t \oplus b_{49}^t \oplus b_{79}^t \oplus b_{84}^t \oplus b_3^t b_{59}^t \oplus b_{10}^t b_{12}^t \oplus b_{15}^t b_{16}^t \\ & \oplus b_{25}^t b_{53}^t \oplus b_{35}^t b_{42}^t \oplus b_{55}^t b_{58}^t \oplus b_{60}^t b_{74}^t \oplus b_{20}^t b_{22}^t b_{23}^t \\ & \oplus b_{62}^t b_{68}^t b_{72}^t \oplus b_{77}^t b_{80}^t b_{81}^t b_{83}^t. \end{aligned}$$

LFSR. The update function $F(L^t)$ used in LFSR is defined as below: it is based on a primitive polynomial over $GF(2)$ and hence always ensures a period of $2^{69} - 1$.

$$F(L^t) = l_0^t \oplus l_5^t \oplus l_{12}^t \oplus l_{22}^t \oplus l_{28}^t \oplus l_{37}^t \oplus l_{45}^t \oplus l_{58}^t.$$

2.2 Initialization Phase

Denote the 128-bit secret key by $k = (k_0, \ldots, k_{127})$ and the 128-bit initial value by $IV = (IV_0, \ldots, IV_{127})$. In addition, there will be extra 22-bit padding denoted by $PD = (PD_0, \ldots, PD_{21}) = 1^{22}$ (all one string). The two registers are initialized based on Equations (1) and (2), respectively.

$$b_i^0 \quad = \quad IV_i \quad (0 \le i \le 89) \tag{1}$$

$$l_i^0 = \begin{cases} IV_{i+90} & (0 \le i \le 37) \\ PD_{i-38} & (38 \le i \le 59) \\ 0 & (60 \le i \le 68) \end{cases} \tag{2}$$

After the two registers are initialized, the state at clock t $(0 \le t \le 510)$ is updated as follows:

$$\begin{aligned} z^t &= O(B^t, L^t), \\ cnt &= l_{62}^t || l_{63}^t || l_{64}^t || l_{65}^t || l_{66}^t || l_{67}^t || l_{68}^t, \\ b_{89}^{t+1} &= G(B^t) \oplus l_0^t \oplus k_{cnt} \oplus z^t, \\ b_i^{t+1} &= b_{i+1}^t \quad (0 \le i \le 88), \\ l_{59}^{t+1} &= F(L^t) \oplus z^t, \\ l_i^{t+1} &= l_{i+1}^t \quad (0 \le i \le 58), \\ l_{i+60}^{t+1} &= ((t+1) >> (8-i)) \wedge 1 \quad (0 \le i \le 8). \end{aligned}$$

The corresponding illustration can be found at Figure 2. Thus in the initialization phase the LFSR is partitioned as two 60 and 9-bit registers. While the 60-bit part is updated using the LFSR logic, the 9-bit part operates as a decimal up-counter. Since the last 7 bits of the LFSR when interpreted as a decimal number (denoted by *cnt*) also forms the index of the key bit used in the NFSR update, this ensures that all the key bits influence the initialization function. Note that $l_{i+60}^{t+1} = ((t+1) >> (8-i))$ for $i \in [0, 8]$ simply means that in the $(t+1)^{th}$ cycle the LFSR bits 60 to 68 are updated with the 8 bits of the decimal up-counter $t+1$. The 60th LFSR location gets the MSB of the counter and 68th location gets the LSB.

2.3 Keystream Generation

After the initialization phase, the state in NFSR becomes $B^{511} = (b_0^{511}, \cdots, b_{89}^{511})$ and the state in LFSR becomes $L^{511} = (l_0^{511}, \cdots, l_{68}^{511})$. Note that since the last 9 LFSR bits worked as a decimal up-counter during the initialization phase, and since the initialization phase

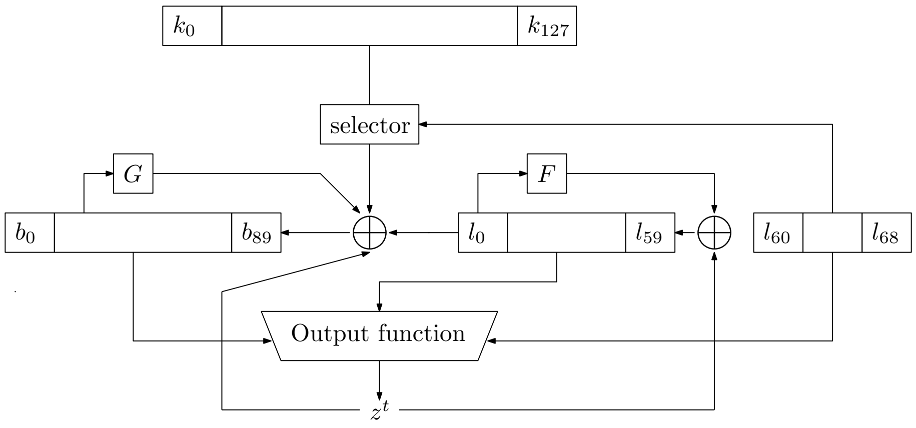

Figure 2: Illustration of the initialization phase

had 511 rounds, the last 9 bits of the LFSR at the beginning of the keystream generation phase will always be the all 1 vector. The first keystream used for plaintext encryption is z^{511}. At the keystream phase, the state at clock t ($t \geq 511$) is updated as follows and the keystream bit z^t will be output. Note that we limit the amount of keystream extractable from a single key-IV pair to 2^{64} bits, which should be sufficient for most applications.

$$\begin{aligned}
z^t &= O(B^t, L^t), \\
cnt &= l_{62}^t || l_{63}^t || l_{64}^t || l_{65}^t || l_{66}^t || l_{67}^t || l_{68}^t, \\
b_{89}^{t+1} &= G(B^t) \oplus l_0^t \oplus k_{cnt} \oplus k_{t\%128}, \\
b_i^{t+1} &= b_{i+1}^t \quad (0 \leq i \leq 88), \\
l_{68}^{t+1} &= F(L^t) \\
l_i^{t+1} &= l_{i+1}^t \quad (0 \leq i \leq 67).
\end{aligned}$$

The corresponding illustration can be found at Figure 3. The circuit level novelty that we were referring to the previous section is actually the additional key filter k_{cnt} that we use. As we have seen earlier, the *cnt* sequence is derived from interpreting the last 7 bits of the LFSR as a decimal number. During the initialization phase it is simply the i mod 128 sequence with i ranging from 0 to 510. However during the keystream generation phase the *cnt* sequence depends on the evolution of the LFSR. Since the LFSR has a period of $2^{69} - 1$, so does the *cnt* sequence. This effectively garbles the order of key bits that is used to update the NFSR. Along with the $k_{t\%128}$ filter, this not only guarantees a high period of the keystream, but as we will show in the following sections, it also provides immunity against many known cryptanalytic techniques used to attack small state stream ciphers.

3 Design Rationale: Use of Double Key Filter

Every design decision taken on a macro level has been predicated by the need to prevent previous known attacks on short state stream ciphers. In this section we try to reason why we took some of the steps to adopt the current algebraic structure of the cipher. Our aim is to clearly establish what the design challenges were and how we attempted to address them. Note that most of the attacks against Lizard leveraged on the fact that the Key-IV mixing function was not injective. This is not the case for Atom, and so we look to prevent other attacks reported in literature.

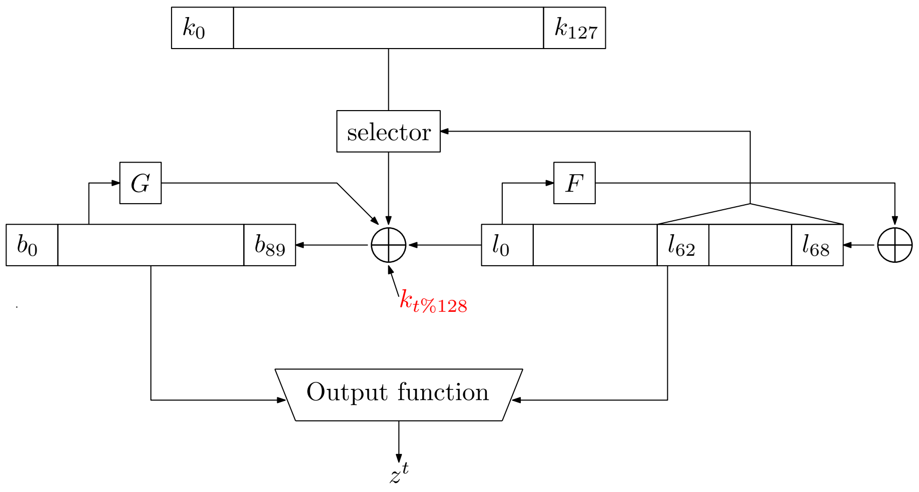

Figure 3: Illustration of the keystream generation phase

3.1 Preventing Banik's Key-Recovery Attack Against Sprout [Ban15]

In [Ban15], it was first pointed out that there exist around 2^{30} IVs for every key in Sprout such that the LFSR becomes all 0 after the Key-IV mixing phase. In the keystream phase the LFSR does not receive feedback from the output bit and hence remains in the zero state throughout the evolution of the cipher, which made the algebraic structure of the cipher weak. The work in [Ban15] leveraged this fact to report the following:

- Key-IV pairs were found in practical time that produced keystream bits with period equal to 80.
- A key recovery attack was reported in the multiple IV mode. The attacker queries keystream for a fixed secret key and multiple secret IVs and waits till an IV is queried such that the LFSR falls into the all 0 state after the key-IV mixing. Once this happens key recovery was shown to be very efficient, since the algebraic structure of the cipher weakened, simple equations on the key bits could be obtained which could be solved to find the secret key.

The zero state problem was tackled quite elegantly in the design of Plantlet. In the key-IV mixing phase Plantlet fixes the final LFSR bit to 1, so that when the cipher finally enters the keystream generation mode the LFSR state is never all zero on the account of the fact that the final bit is 1. Since the Plantlet LFSR connection polynomial is primitive and has length 61, it has a period of $2^{61}-1$ and it never falls into the all zero state. This effectively prevents all the above attacks. In Atom, the solution we adopt is indeed inspired by Plantlet. During the key-IV mixing phase the last 9 bits of the LFSR effectively work as a decimal counter. Since the number of rounds in the initialization phase is 511, the last 9 bits of the LFSR are all 1 when we enter the keystream generation phase. The Atom LFSR connection polynomial is also primitive guaranteeing a period of $2^{69}-1$ and the fact that it never falls into the all 0 trap.

In the keystream generation phase the keybit-sum to update the NFSR is $k_{cnt} \oplus k_{t\%128}$. *cnt* is basically the decimal number formed by interpreting the last 7 LFSR bits as a decimal number. Since the LFSR has a period of $2^{69}-1$, it is to be expected that the *cnt* sequence would also have the same period. Therefore it is clear that both the *cnt* and $t\%128$ sequence taken together would repeat only after $lcm(2^{69}-1, 128) = 2^{76}-128$ clock cycles. This also guarantees that the Atom keystream, produced by most key-IV pairs, has a minimum period of $2^{76}-128$ bits (unless we have some degenerate cases like the all one/zero key for which the minimum period is given by the period of the LFSR i.e. $2^{69} - 1$).

3.2 Preventing Banik-Barooti-Isobe like Attacks Against Plantlet [BBI19]

Before we outline how our design prevents the attack outlined in [BBI19] against Plantlet, it would be best to summarize the attack in a few words:

- The attackers observe that if two internal states of Plantlet differ by $0^{40}||0^{42}||1||0^{18}$, i.e. the 43rd LFSR bit, then they produce keystream vectors whose difference is 0 with probability 1 in 41 clock cycles. The difference is 1 also with probability 1 in 4 other clock cycles.

- They try to obtain two internal states with single bit difference in the 43rd LFSR location by querying some fixed key and random IVs. The keystream extractable with a single key-IV pair is limited to 2^{30} bits. It was calculated that the probability that such difference in states will occur during the course of a single key-IV query is around $2^{-54.6}$. Hence to get the required difference at least once on average the attackers needed to query around $2^{54.6}$ different IVs.

- They inspect the keystream blocks extracted with the $2^{54.6}$ IVs. They can with some probability guess that when 2 segments of keystream blocks possess the 45 bit difference just mentioned, they have been produced by two internal states that differ only in the 43rd LFSR location.

- Thereafter by solving a system of polynomial equations given by the keystream bits, the attacker can find the secret key if his guess was indeed correct, or reach some kind of contradiction if his guess was incorrect. In the latter event, they repeat the procedure for other keystream blocks with the given difference. The process when repeated a finite number of times, does indeed yield the value of the secret key.

For Atom, we experimented with differences of up to hamming weight 4. The best differential characteristic was obtained when 2 Atom states differ at the 55th LFSR location: they are guaranteed to produce keystream that are either equal or unequal in only 18 clock cycles with probability 1. This number is quite small, compared to Plantlet where 45 keystream bits are guaranteed to be equal/unequal for a well chosen difference in the LFSR state. The reason for this is that, in Atom because of the fact that the last 7 bits of the LFSR provide the index of the key bit which is used to update the NFSR. Thus any LFSR difference will after a short amount of time be fed back into the 68th LFSR location through the LFSR taps. When this happens different key bits are used to update the NFSR and the so a difference is propagated into the NFSR relatively quickly.

In any case, when we generate N keystream bits, the probability that there exists 2 clock instances t_1, t_2 at which the internal states differs only in the 55th LFSR bit is given by birthday bound considerations and is around $p = N^2 \cdot 2^{-160}$. For $N = 2^{64}$, this probability is around 2^{-32}. The attacker would therefore need to query around $V = p^{-1} = N^{-2} \cdot 2^{160}$ IVs to get one difference state on average. The attacker then filters out all internal state pairs for which the keystream segment does not produce the 18 bit difference pattern. The number of such pairs is around $U = V \cdot \frac{(N)(N-1)}{2} \cdot 2^{-18} \approx 2^{141}$. The attacker would have to then solve polynomial systems arising from these 2^{141} state pairs. This is well above the complexity of exhaustive search. Note that the main reason that the attack is prevented is that the use of k_{cnt} term used in the state update depending on the last 7 LFSR bits. This prevents us from getting a large enough differential pattern of equal/unequal bits that can bring down the value of U to less than 2^{128}.

3.3 Preventing Todo-Meier-Aoki like Attacks Against Plantlet [TMA19]

The attacks reported in [TMA19] are against an instance of Plantlet that allows up to 2^{54} keystream bits per key-IV pair and are correlation attacks of a similar type reported against the Grain family [BGM06, TIM+18]. The attack can be summarized as follows:

- The attackers try to formulate probabilistic equations of the following form:

$$\bigoplus_{t\in\mathbb{T}_Z} z_t \oplus \bigoplus_{t\in\mathbb{T}_K} k_t \oplus \bigoplus_{t\in\mathbb{T}_L} l_i^t = \epsilon \tag{3}$$

  where $\mathbb{T}_Z,\ \mathbb{T}_K,\ \mathbb{T}_L$ are linear masks that denote subsets of keystream, key and LFSR state bits that add up to form the probabilistic equations. ϵ denotes a variable such that $Pr[\epsilon = 0] = 1/2 \cdot (1+\eta)$, where η is the correlation term whose value determines the complexity of the attack algorithm.

- The value of η depends on the linear biases of the NFSR update function and the output function.

- In Plantlet, the keybit used in the state update function repeats every 80 clock cycles. To apply the correlation attack to Plantlet, the attackers need $\bigoplus_{t\in\mathbb{T}_K} k_t$ to be a constant. Therefore the attackers build up a bank of equations at intervals of 80 cycles each. For example if the original probabilistic equation was $z_t \oplus z_{t+2} \oplus k_t \oplus k_{t+45} \oplus k_{t+51} \oplus l_{10}^t + l_{14}^t \oplus l_{31}^t = \epsilon$, the attackers build equation banks of the form:

$$\begin{aligned}
z_0 \oplus z_2 \oplus k_0 \oplus k_{45} \oplus k_{51} \oplus l_{10}^0 + l_{14}^0 \oplus l_{31}^0 &= \epsilon_1\\
z_{80} \oplus z_{82} \oplus k_0 \oplus k_{45} \oplus k_{51} \oplus l_{10}^{80} + l_{14}^{80} \oplus l_{31}^{80} &= \epsilon_2\\
z_{160} \oplus z_{162} \oplus k_0 \oplus k_{45} \oplus k_{51} \oplus l_{10}^{160} + l_{14}^{160} \oplus l_{31}^{160} &= \epsilon_3\\
\vdots &\ \vdots
\end{aligned}$$

  Note that the value of $\bigoplus_{t\in\mathbb{T}_K} k_t$ remains constant in these equations because in Plantlet the keybits used in the state update repeat every 80 cycles.

- Once the attacker has around N such equations and the correlation of each ϵ_i is η, then for the correct value of the LFSR internal state the difference between the number of correct and incorrect equations will be distributed according to $\mathcal{N}(N\eta, N\eta^2)$ if $\bigoplus_{t\in\mathbb{T}_K} k_t = 0$ or $\mathcal{N}(-N\eta, N\eta^2)$ if $\bigoplus_{t\in\mathbb{T}_K} k_t = 1$. This distribution when the LFSR internal state is incorrect is given by $\mathcal{N}(0, N\eta^2)$, where $\mathcal{N}(\mu,\sigma)$ denotes the normal distribution with mean μ and variance σ. Note that around $N = O(\eta^2)$ equations are needed to reliably distinguish the distributions $\mathcal{N}(\pm N\eta, N\eta^2)$ and $\mathcal{N}(0, N\eta^2)$ and mount the attack.

- The authors use a maximum likelihood decoding algorithm like the Fast Walsh Hadamard transform (FHWT) to find the LFSR state efficiently. Thereafter the key and NFSR state can be found by solving polynomial equations on the keystream bits (see [BGM06]).

The reason why this attack can not be used against Atom is as follows. In Atom we can also derive probabilistic equations as given in Equation (3). However, the key bits used in the state update depend on both the LFSR and the time counter and as such is not known completely to the attacker. Since the LFSR is guaranteed to begin the keystream phase with a non zero state and that its connection polynomial is primitive over $GF(2)$, it will only repeat after $2^{69} - 1$ cycles. The mod 128 counter repeats only after 128 cycles. Hence the keybits repeat only after $\tau = lcm(2^{69} - 1, 128) = 2^{76} - 128$

clock cycles. This greatly increases the data complexity of any correlation attack on Atom. Since we limit the amount of keystream to 2^{64} per key-IV pair, this type of attack seems infeasible. Nevertheless, we heuristically searched different linear combinations to mount a linear attack. However, we did not find any efficient linear combination of keystream bits that has a high enough correlation with the sum of LFSR bits and keybits. We give one such example in Appendix C.

3.3.1 When less than the full LFSR sequence repeats

So far our arguments for security have relied on the fact that the entire LFSR must take $P = 2^{69} - 1$ iterations before it repeats. However, note that if we are able to formulate an attack that requires only a part of the LFSR sequence to repeat then it may take less than P iterations. First of all, let us try to answer the question: when can a part of the LFSR sequence repeat? We are primarily interested in the situation when the sequence *cnt* denoted by the last 7 bits of the LFSR repeats. To simplify notations denote by $cnt_t = l^t_{62}||l^t_{63}||l^t_{64}||l^t_{65}||l^t_{66}||l^t_{67}||l^t_{68}$ to be the value of *cnt* at any instant t of the keystream generation phase. We are interested in finding whether $cnt_{t+i} = cnt_{t+T+i}$ for $i = 0, 1, \ldots, r-1$, for some particular value of $T > 7$. In other words, we want to see if the sequence *cnt* repeats for r successive iterations exactly T iterations apart. We already know that if $T = P$, then the above will hold for any value of r: however, if $T < P$, then the above may hold for smaller values of $r < 63$. For any arbitrary value of $T < P$, $cnt_{t+i} = cnt_{t+T+i}$ for $i = 0, 1, \ldots, r-1$, is system of $r + 6$ linear equations in the 69 LFSR variables $X = l^t_0, l^t_1, \ldots, l^t_{68}$. In matrix notation this can be written as $L_T X = \mathbf{0}$, where L_T is a $(r + 6) \times 69$ matrix over GF(2) which of course depends on the exact value of T. The rank of L_T is at most $r + 6$ and so the above equation can have at least 2^{63-r} solutions. Discarding the all zero state, and assuming that all other LFSR states occur equally during the keystream generation, the probability that there is a repetition for r cycles exactly T iterations apart is around $\frac{2^{63-r}-1}{2^{69}-1} \approx 2^{-r-6}$.

Now let us get back to the example we looked at in the previous subsection. Let us say we have a probabilistic equation of the form

$$Eq_t : \bigoplus_{t \in \mathbb{T}_Z} z_t \;\oplus\; \bigoplus_{t \in \mathbb{T}_K} k_t \;\oplus\; \bigoplus_{t \in \mathbb{T}_L} l^t_i = \epsilon.$$

Note that the attacker does not know the value of the internal bits l^t_i of the state of Atom. In order for the attacker to continue the correlation attack, he needs the value of the sum of keybits $\bigoplus_{t \in \mathbb{T}_K} k_t$ to be a constant in the system of equations to apply a method like the FHWT to recover the LFSR state. We have already seen that if the correlation in each probabilistic equation is η then it requires $N = O(\eta^{-2})$ equations to successfully mount the attack. Let us say that the attacker chooses $T_1, T_2, \ldots, T_N \equiv 0 \bmod 128$, to generate the N equations required for the attack i.e. the equations he chooses are

$$\bigoplus_{t \in \mathbb{T}_Z} z_{t+T_j} \;\oplus\; \bigoplus_{t \in \mathbb{T}_K} k_{t+T_j} \;\oplus\; \bigoplus_{t \in \mathbb{T}_L} l^{t+T_j}_i = \epsilon$$

for $j \in [1, N]$. Since the attacker does not know the internal state of Atom, he can not do better than guess the T_j's randomly. The probability that the sum of keybits $\bigoplus_{t \in \mathbb{T}_K} k_{t+T_j}$ are constant is

- $2^{-(r+6) \cdot N}$, if the values T_j are selected so that $cnt_t = cnt_{t+T_j}$ for all j, and
- 2^{-N}, if the above does not hold.

In the above example r is the number of successive values of *cnt* used to formulate each equation Eq_t. Its value is not immediately deduced from the algebraic form of Eq_t

but would depend on the internal structure of the cipher. Summing the above probability with Bayes theorem, the above comes to around $2^{-(r+6)\cdot N} + (1 - 2^{-(r+6)\cdot N}) \cdot 2^{-N} \approx 2^{-N}$. Since the attacker is not aware of the internal state, he can do no better than collect N randomly selected equations of the above form, and hope that the sum of keybits are constant in these equations. The probability that he succeeds is exponentially low.

3.4 Preventing Esgin-Kara like Attacks Against Sprout [EK15]

One of the reasons Sprout was immune to the TMD Tradeoff attacks of [BS00], (despite having state size equal to the size of the key), was that the key was used to update the state update transitions, and so it was impossible to construct a function that mapped state to any length of keystream prefix, without using the key. Thus the effective size of the internal state was twice the size of the key as required. The attack against Sprout proposed in [EK15] can be summarized as follows:

- The authors showed that in spite of all the above, there existed special states of Sprout for which it is possible to map the state to keystream without requiring the secret key. This is because the round key function used to update the state at time t was of the form $(k[t \bmod 80]) \cdot (\sum n_i^t + \sum l_i^t)$, where $k[i]$ is the i-th key bit. Now if the expression $(\sum n_i^t + \sum l_i^t)$ is 0 for 40 consecutive cycles then the contribution of the key to the state update is 0, which in other words means that it is now possible for these special states to produce keystream without requiring the key and so a function mapping state to a 40-bit keystream segment is now possible, which does not additionally require the secret key.

- The authors showed that it was possible to efficiently enumerate these special states and tabulate these states along with the 40 bit keystream segment produced by them.

- In the online stage of the attack, the authors examined every available keystream segment and look for the segment in the precomputed tables and try to extract the state. It was shown that if done sufficient number of times, Sprout will degenerate into a special state and the attacker can extract the corresponding state from the table. After this the attacker solves some algebraic equations to find the secret key.

- The authors showed that both the online and offline stages of the attack had practical time, memory and data complexities.

Ever since [EK15] was published, all subsequent stream cipher designs like Plantlet and Lizard have been proposed with strictly linear key bit contributions to the state update function, i.e. of the form $k[t \bmod \text{keysize}]$. We too adopt linear key addition to the state update, but the key bit to be used is not indexed by a counter but by the last 7 bits of the LFSR. This gives rise to some unique opportunities. To understand better, let us consider an altered variant of Atom in which the state update function does not include the $k_{t\%128}$ term, i.e.

$$\begin{aligned} b_{89}^{t+1} &= G(B^t) \oplus l_0^t \oplus k_{cnt}, \\ l_{68}^{t+1} &= F(L^t). \end{aligned}$$

Note that the LFSR runs autonomously during keystream generation, and since initialization lasts 511 cycles, the cipher always enters keystream phase with last 9 bits all 1. Since the connection polynomial is primitive, it is well known, that the LFSR has a period of $2^{69} - 1$ and never enters the all zero state. Now let us say that for some t, the last 7 bits of the LFSR is 000 0001 (in hex `01`) which occurs with probability 2^{-7}.

We want that for T consecutive cycles the last 7 bits of the LFSR take the following values 01,02,04,08,10,20,40,01... and so on. This way the only keybits involved in the update of the NFSR state will be from the 7-element set $\mathbb{K} = \{k_1, k_2, k_4, k_8, k_{16}, k_{32}, k_{64}\}$. For this to happen the following equations to hold

1. $l_{62}^t, l_{63}^t, l_{64}^t, l_{65}^t, l_{66}^t, l_{67}^t, l_{68}^t = [000\ 0001]$

2. $l_{68}^{t+i} = F(L^{t+i-1}) = 0$ for $i = 1, 2, 3, 4, 5, 6 \bmod 7$ and $1 \le i < T$

3. $l_{68}^{t+i} = F(L^{t+i-1}) = 1$ for $i = 0 \bmod 7$ and $1 \le i < T$.

There are a total of $T + 6$ conditions to fulfill and it is not difficult to see that the solution space of the above linear system of equations has dimension $69 - (T + 6) = 63 - T$. Since the system is linear it is easily possible to enumerate all such states. Note that for all such 2^{63-T} LFSR states, we could easily define a function that maps the entire internal state + $\mathbb{K}$ to a T-bit keystream segment. So our attack would be as follows:

Offline:

A: For all 2^{90} NFSR states, 2^{63-T} LFSR states enumerated above and all 2^7 values of $\mathbb{K}$ (total 2^{160-T} iterations)

[1:] Compute the T bit keystream segment Z produced by it.

[2:] Store in a Table $Tab[Z]$= State|| 7 Keybits used to produce it.

[3:] Note that each table cell will store around 2^{160-2T} entries (a total of 2^{160-T} entries are divided among 2^T table cells).

B: Time complexity = 2^{160-T}, Memory complexity = $160 \cdot 2^{160-T}$

Online:

A: Take any keystream segment Z.

B: Extract all 2^{160-2T} entries S from $Tab[Z]$

C: For all such S:

[1:] Solve for the keybits not in $\mathbb{K}$.

[2:] If a solution exists then output key else try another value of S.

Let us try to find the complexity of the online phase. Note that we have stored a fraction $1/2^{T+6}$ of all internals states. Under standard randomness assumptions, the online stage will extract the correct state in $2^{T+6} \cdot 2^7$ attempts when both the state and key are correctly extracted. Hence the complexity of the online phase is $2^{T+13+160-2T} = 2^{173-T}$ attempts to solve for the remainder of keybits. Assuming that finding the rest of the key bits is efficient, taking $T \approx 60$, would put the complexity of both the online and offline phase below the complexity of exhaustive search.

Now when we include the $k_{t \bmod 128}$ term in the update function it ensures that we can never get a relation between the state and keystream segment that only involves a few key bits. More precisely any function mapping into $T \le 128$ keystream bits must involve at least $159 + T$ input bits of the state and key. Thus this attack is prevented.

Replacing the Decimal Counter: For the purpose of saving area, one can probably consider replacing the decimal counter by a simpler register like an LFSR. Consider what happens when we do this: for example, let us say we have a 9-bit primitive LFSR $d_0, d_1, \ldots, d_8$, which updates as $d_0, d_1, \ldots, d_8 \rightarrow d_1, d_2, \ldots, d_0 \oplus d_4$, or the 9-bit maximum length NFSR $d_0, d_1, \ldots, d_8 \rightarrow d_1, d_2, \ldots, 1 \oplus d_0 \oplus d_4 \oplus \prod_{i=1}^{8} d_i$. Let dec_t be the integer formed at time t by the last 7 bits $d_2, d_3, \ldots, d_8$. If this register replaces the decimal counter, then the contribution of the keybits to the update function becomes $k_{cnt_t} \oplus k_{dec_t}$, in place of $k_{cnt_t} \oplus k_{t\%128}$. Then it is possible to have $T = 60$ consecutive clock cycles in which only as less as 42 keybits are involved in the state update. This is because the sequence $dec_t, dec_{t+1}, \ldots, dec_{t+59}$ can have as low as 42 separate values for some t for both the LFSR and the maximum length NFSR. Then we would no longer be able to argue that "any function mapping into $T \leq 128$ keystream bits must involve at least $159 + T$ input bits of the state and key". To counter this problem, we could instead use a 7 bit register (and interpret dec_t to be the integer formed by the entire state of the counter). But such a register can not count up to 511 which is the required number of initialization rounds. So to overcome this we would need additional logic to count upto 511 which makes the total gate count quite close to an ordinary decimal counter.

Note that including the $k_{t\%128}$ term in the update function also automatically prevents the attack of [ZGM17]. The attack in [ZGM17] was an extension of [EK15], in the sense that the authors used an additional property of the output Boolean function called k-normality to simplify the construction of special states. A function is called k-normal if it is constant on a k-dimensional subspace of its input bits. Using this property, the attack further refines the definition of special states to mean those for which the output function is evaluated in the k-dimensional space for a given number of rounds, which are again listed in tables. Using this technique the TMD tradeoff attack they propose is around 2^{10} times faster than [EK15]. Again, since Atom does not use non-linear mixing, this attack is not applicable.

4 Security Evaluation

4.1 TMD Tradeoff Attacks

TMD tradeoff attacks aim to invert a one way function f at a single point in the range of function. The attack is probabilistic and the attacker may need access to multiple points in the range of f. For stream ciphers, the one way function is typically the map between the internal state and the prefix of the keystream bits produced by the internal state. We have already seen in the previous section that any function mapping into $T \leq 128$ keystream bits must involve at least $159 + T$ input bits of the state and key. In this context, let us look at some of the common TMD Tradeoff attacks reported against stream ciphers:

Biryukov-Shamir Attack [BS00]: Given that N is the cardinality of the set of internal states, the attacker chooses m, t, D so that $mt^2 = N$ and $t \geq D$, where D will be the data complexity required for the attack. The attacker builds $\frac{t}{D}$ tables of size $m \times t$ in the following manner: he randomly chooses m initial states. For each initial state, he forms a chain of length t by iteratively applying the stream cipher function f and using the keystream as the state for the next point. For each table some unique reordering of the bits after applying the function f is used so that the tables do not store the same set of states. For the attacker to be able to do this he must be able to formulate the stream cipher function in such a manner that it maps equal length input and output bit vectors. The only choice the attacker has is to choose $T = 159 + 128 = 287$, which enables him to formulate the function f as mapping the key and state to the first 287 keystream bits produced by the generator.

In the process, $mt \cdot \frac{t}{D} = \frac{N}{D}$ of the state space is covered by all the chains. This also happens to be the offline complexity $T_{offline}$ of this stage. Also only the start and endpoints of each chain are stored in tables, and so $M = m \cdot \frac{t}{D}$ bits of memory is used.

In the online phase, the attacker has access to D segments of keystream. For each target keystream segment y, he applies f on y upto t times checks if y is present as an endpoint in any table. If yes, he goes back to the starting point and retrieves the state just before y in the chain. The total time complexity is thus $T_{online} = D \cdot t \cdot \frac{t}{D} = t^2$. This gives us the tradeoff curve $T_{online} M^2 D^2 = N^2$, with the limitation that $T_{online} \geq D^2$. For Atom, $N = 2^{287}$ and there is no point in the tradeoff curve for which T_{online} and $T_{offline}$ are both less than 2^{128}.

Using Sampling Resistance of Atom: The main idea of sampling is to find an efficient way to generate and enumerate "special" cipher states, for which the first few keystream bits generated by the cipher is a fixed string. If this happens for a run of x bits, the sampling resistance of the cipher is defined to be $R = 2^{-x}$. This leads to improved trade-off attacks when the value of x is significant.

For Atom, the taps are extremely densely packed. The sampling resistance is quite large and around 2^{-5}. Observe the following set of equations in the generator:

$$\begin{aligned}
b^t_{22} &= z_t \oplus b^t_1 \oplus b^t_5 \oplus b^t_{11} \oplus b^t_{36} \oplus b^t_{53} \oplus b^t_{72} \oplus b^t_{80} \oplus b^t_{84} \\
&\quad \oplus l^t_5 l^t_{16} \oplus l^t_{13} l^t_{15} \oplus l^t_{30} l^t_{42} \oplus l^t_{22} l^t_{67} \oplus h(l^t_7, l^t_{33}, l^t_{38}, l^t_{50}, l^t_{59}, l^t_{62}, b^t_{85}, b^t_{41}, b^t_9) \\
b^t_{23} &= z_{t+1} \oplus b^t_2 \oplus b^t_6 \oplus b^t_{12} \oplus b^t_{37} \oplus b^t_{54} \oplus b^t_{73} \oplus b^t_{81} \oplus b^t_{85} \\
&\quad \oplus l^t_6 l^t_{17} \oplus l^t_{14} l^t_{16} \oplus l^t_{31} l^t_{43} \oplus l^t_{23} l^t_{68} \oplus h(l^t_8, l^t_{34}, l^t_{39}, l^t_{51}, l^t_{60}, l^t_{63}, b^t_{86}, b^t_{42}, b^t_{10}) \\
&\;\;\vdots \\
b^t_{26} &= z_{t+4} \oplus b^t_5 \oplus b^t_9 \oplus b^t_{15} \oplus b^t_{40} \oplus b^t_{57} \oplus b^t_{76} \oplus b^t_{84} \oplus b^t_{88} \\
&\quad \oplus l^t_9 l^t_{20} \oplus l^t_{17} l^t_{19} \oplus l^t_{34} l^t_{46} \oplus l^t_{26} l^t_{71} \oplus h(l^t_{11}, l^t_{37}, l^t_{42}, l^t_{54}, l^t_{64}, l^t_{66}, b^t_{89}, b^t_{45}, b^t_{13})
\end{aligned}$$

This means that given the value of 282 particular state and key bits of Atom and the first 5 keystream bits produced from that state, another 5 internal state bits may be deduced efficiently. The equation for b^t_{27} involves b^t_{90} which already contains the b^t_{25} term.

This helps us define a function $f : \{0,1\}^{282} \to \{0,1\}^{282}$. We fix a specific 5 bit string say 1^5. For any 282-bit string x we expand it to a 287-bit string by interpreting it as a partial state of Atom and calculating the remaining 5 bits $b^t_{22}, b^t_{23}, \ldots, b^t_{26}$ by assuming that $z_t, z_{t+1}, \ldots, z_{t+4} = 1^5$. Generate the remaining 282 bits $y = z_{t+5}, z_{t+6}, \ldots, z_{t+286}$ by clocking the Atom generator with the full state. We define $f(x) = y$. Using this technique, in the online stage we wait till we observe the 1^5 vector in the keystream sequence. If so we try to invert f using the subsequent 282 bits of the keystream.

It can be shown that the tradeoff curve resulting thereof is $T_{online} M^2 (RD)^2 = (RN)^2$, with the condition $(RD)^2 \leq T$. Again there is no point in the tradeoff curve for which T_{online} and $T_{offline}$ are both less than 2^{128}.

Hong-Sarkar Attack [HS05]: This attack is exactly same as [BS00], except that the definition of the underlying one way function is now changed. In this attack f maps the string consisting of the Key and IV to an equal length keystream bits. Thus if K and V refer to the size of the Key space and IV space respectively, then $N = KV$, and we will have the new tradeoff curve $T_{online} M^2 D^2 = K^2 V^2$ with the limitation that $T_{online} \geq D^2$. This attack becomes applicable if $V \ll K$, as in the case of the A5/3 cipher (in which the size of the secret key in 64 bits, and the size of the IV is 22 bits). In our case $K = V$, and $N = KV = 2^{256}$, and again there is no point in the tradeoff curve for which T_{online} and $T_{offline}$ are both less than 2^{128}.

Dunkelman-Keller Attack [DK08]: The Dunkelman-Keller TMD attack is a multiple IV attack, i.e. the attacker obtains keystreams from multiple IVs and the same Key in order to perform the attack. The definition of the underlying one way function f is slightly different from the Hong-Sarkar attack. Given a fixed IV, the function f maps the secret key to the keystream sequence of equal length. The attacker chooses $\frac{V}{D}$ random IVs. For each IV he constructs t tables as before, by iterative application of the function f from m random starting points, with $mt^2 = K$. Again only the start and end points are stored and so for each IV the storage required is $M_{single} = mt$, and the total storage is therefore $M_{single} \cdot \frac{V}{D}$, and the total offline complexity is $T_{offline} = K \cdot \frac{V}{D}$.

In the online phase, the attacker waits until he receives keystream for one of the $\frac{V}{D}$ IVs he had made tables for. This happens in roughly D IV resynchronizations. Once he gets such keystream from such an IV, he retrieves the t tables he had constructed for the particular IV and tries to find the inverse image of the keystream string in each of the tables. Therefore the online complexity is given by $T_{online} = D + t^2 = D + \frac{K^2}{M^2_{single}} = D + \frac{K^2V^2}{M^2D^2}$ with the constraints $T_{online} \geq D, V \geq D$. In the case of Atom, $KV = 2^{256}$ and again this attack is not feasible.

A note about amortization of $T_{offline}$: So far, our security argument has heavily relied on the fact that the total cost of any attack in terms of computational complexity, including the time spent on building offline tables can not be smaller than the complexity of exhaustive search. However, it is possible to amortize the complexity of $T_{offline}$ over many cryptanalysis attempts. For example consider the tradeoff curve for Atom for the Biryukov-Shamir Attack: $T_{online}M^2D^2 = N^2$, with $T_{online} \geq D^2$ and $T_{offline} = \frac{N}{D}$. A point on this curve is $M = 2^{166},\ D = 2^{60},\ T_{offline} = 2^{227}$ with $T_{online} = 2^{122}$. Note in this point we have brought down the online complexity of the attack below exhaustive search, though the offline complexity is many times more. Note that the complexity of offline tables etc is a one-time cost. So the same tables can be used to attack the cipher multiple times. Consider the situation when we try to recover the key for 2^{100} independently generated keystreams. The total complexity for this using the TMD approach is $2^{100} \cdot T_{online} + T_{offline} \approx 2^{227} < 2^{100} \cdot 2^{128}$. This is less than the time complexity of 2^{100} exhaustive searches, and in a sense we have managed to amortize/average out the higher complexity of $T_{offline}$ over multiple attacks. This is exactly the approach attempted in some papers [Bjo08, HK05]. The cryptographical community seems divided over this issue: while some papers see amortization of the offline complexity as a legitimate attack approach, whereas [BS00], for example, consider the time taken to mount one attack representative of the true security level of the design (in fact the "state size should equal at least twice the size of key" argument was posited in [BS00] when considering a single attack). In this paper, we argue the security of our design in the context of a single attack.

4.2 Differential Attack

Differential cryptanalysis was first introduced to analyze the block ciphers DES and Feal and the hash function N-Hash by Biham and Shamir [BS90, BS91]. Since then, it has been applied to many symmetric cryptographic primitives, not limited to block ciphers. To evaluate the resistance against differential attacks for block ciphers, one way is to obtain the lower bound of the number of active S-boxes, which is a nonlinear operation. For Atom, there are two nonlinear components of NFSR and output function h, which include AND operations as nonlinear operation. Therefore, instead of an active S-box, we search the lower bound of the number of sum of active AND and $h(X)$, which means having at least 1 active bit as input. In our evaluation, since the maximum differential probability of AND is 2^{-1} and $h(X)$ consists of AND and XOR, we count the maximum differential probability of $h(X)$ as 2^{-1}. Hence, it is sufficient to guarantee the security against the differential attack if there are 128 sum of active AND and $h(X)$. We present this security evaluation with a MILP-based method [MWGP11], which is well known as the efficient search method to obtain the lower bound of the number of sum of active AND and $h(X)$ (active S-boxes). Our evaluation uses the Gurobi optimizer [Inc15] as a MILP solver, and searches all bit-wise differential characteristics.

Note that our evaluation assumes that each active AND is independent in a differential characteristic. Thus, it might include invalid differential characteristics. However, since our search can cover all valid ones at the same time, we believe that our evaluation is sufficient for obtaining lower bounds of the number of active AND.

Table 1: The lower bound of the sum of AND and $h(X)$ in the related IV setting.

# of rounds	40	45	50	60
# of sum of active AND and $h(X)$	18	24	30	43

Table 1 shows the minimum number of the sum of active AND and $h(X)$ for 40, 45, 50, 60 clocks at the initialization phase in the related IV setting. In our evaluation, we can search it for up to 60 clocks with a computer equipped with 48 cores and 256 GB RAM. Form this result, Atom achieves 128 sum of active AND and $h(X)$ for more than 180 clocks. Thus, we expect that the full rounds of Atom can resist differential cryptanalysis.

4.3 Conditional Differential Cryptanalysis

Conditional differential cryptanalysis was first introduced in [BB93]. The technique allows for improved key recovery and distinguishing attacks against a group of ciphers such as Trivium, KATAN, Grain-v1 and Grain-128. For example, the authors of [KMN11] demonstrated a 961-round distinguisher for Trivium for a large class of weak keys. Attacks against reduced round variants of the Grain family were reported in [KMN10, Ban16].

In its core, conditional differential cryptanalysis is a refinement of ordinary differential attacks where the longevity of differential trails is extended by imposing some conditions on public parameters such as initialization vectors. Denote by $x = (x_1, \dots, x_n) \in \{0,1\}^n$ an n-bit initialization vector and let $\Delta x = (\Delta x_1, \dots, \Delta x_n) \in \{0,1\}^n$ be an IV difference such that $x + \Delta x = (x_1 + \Delta x_1, \dots, x_n + \Delta x_n)$. Furthermore, let $t_i(k, x)$ be the newly generated state bit in round i based on some secret key k and the public parameters x. The IV difference Δx propagates to $t_i(k, x)$ whenever

$$\Delta t_i(k, x) + t_i(k, x + \Delta x) = 1.$$

In order to attain a simple high-round differential trail the attacker can impose conditions on the public parameters to prevent the propagation of the differential into the state in certain rounds. More specifically, these conditions are categorized into two types:

- **Type 1:** Conditions that only involve IV variables, i.e., $w_1(x) \in \{0,1\}$.
- **Type 2:** Conditions that involve both IV and key variables, i.e., $w_2(x,k) \in \{0,1\}$. w_2 should be of the form $w(x,k) = f(x)+g(k)$, where the function $f(x)$ only depends on the IV bits and and the function $g(k)$ only depends on the key bits.

For correctly chosen conditions that prevent the propagation of the differential an adversary hopes to find a biased keystream bit in some round. Then by leveraging this distinguisher the attacker partitions the IV space into 2^N subsets (where N is the total number of type 2 conditions) one for each type 2 condition. A bias should then occur for the one subset for which $g(k)$ is correctly guessed.

The derivation of these conditions can be achieved through computer algebra systems. Evidently, the algebraic expressions of a large number of rounds can be explicitly evaluated when the state update function of the cipher is simple. For example, in Trivium it is effortlessly possible to compute the algebraic equations for more than 200 rounds. This does not hold true for Atom, in fact, due to the complex nature of its state update function, it is not possible to compute the equations of than a dozen rounds, which naturally limits the applicability of conditional differential attacks.

We searched for single-bit differentials heuristically. The best single-bit input differential trail in Atom is found when a difference is introduced in bit IV_{67}. We proceed by stopping its propagation into subsequent rounds with the following conditions:

$$\begin{aligned}
t = 5: &\quad IV_{73} = 0 \\
t = 6: &\quad IV_{81} = 0 \\
t = 7: &\quad IV_{11} = 0 \\
t = 8: &\quad IV_{64} = 0 \\
t = 11: &\quad IV_{70} = 0 \\
t = 13: &\quad IV_{39} = 1 \\
t = 14: &\quad IV_{22} = 0,\ IV_{56} = 1,\ IV_{112} = 0, \\
&\quad f_1(IV \setminus \{IV_0\}) + IV_0 + k_0 = 1, \\
&\quad f_2(IV \setminus \{IV_8\}) + IV_8 + k_1 + k_2 + k_4 + k_5 + k_{10} = 1.
\end{aligned}$$

Here f_1, f_2 are polynomials on both key and IV bits. At this point, the propagation of the differential is prevented during the first 15 rounds. Two of the conditions are of type 2, however the polynomials f_1 and f_2 are infeasible to enumerate unless more IV variables are set to zero, i.e.,

$$IV_i = 0,\, i \in [10, 38] \cup [46, 55] \cup [57, 66] \cup [68, 89] \cup [100, 127]$$

This means that in total 98 IV bits have to bit set to either 0 or 1, which leaves 30 free variables. After this f_1, f_2 become polynomials defined only on the IV bits. We note that IV_0 only occurs linearly in f_1, the same is true for IV_8 in f_2. As we have two type 2 conditions we partition the IV variables into $2^2 = 4$ sets T_U of the form

$$\begin{aligned}
T_U = \{IV \in \{0,1\}^{128} \mid\ & IV_i = 0,\ i \in [10, 38] \cup [46, 55] \cup [57, 66] \cup [68, 89] \cup [100, 127], \\
& IV_{39} = 1,\ IV_{56} = 1, \\
& IV_0 = f_1 + k_0,\ IV_8 = f_2 + k_1 + k_4 + k_5 + k_{10}\},
\end{aligned}$$

where $U = [k_0, k_1 + k_2 + k_4 + k_5 + k_{10}]$. These conditions then produce a detectable bias in the keystream bit of round 36 for the IV set T_U where U has been guessed correctly. We give the full polynomials f_1 and f_2 in Appendix D.

Note a bias may linger longer in the cipher state. For example a difference in IV bit IV_{18} produces a difference in b_{12}^{6} under the conditions

$$IV_{17} = 0, \quad IV_{19} = 0.$$

This conditional differential trail exhibits a bias in state bits b_0^{117} and l_0^{117}, i.e., in round 117. However, it is not clear how such a biased state bit can be exploited given the complicated nature of the keystream function. Consequently, we believe that Atom resists conditional differential attacks with a large security margin.

4.4 Integral/Cube Attacks

The integral attack was first proposed by Daemen et al. as a dedicated attack against the block cipher Square [DKR97], and then it was formalized to the integral property by

Knudsen and Wagner [KW02] (the saturation attack of Lucks [Luc01] also belongs to this family of attacks). We define the four states for a set of 2^n n-bit cell: **A**: if $\forall i, j\ i \neq j \Leftrightarrow x_i \neq x_j$, **C**: if $\forall i, j\ i \neq j \Leftrightarrow x_i = x_j$, **B**: $\bigoplus_i^{2^n-1} x_i$, and **U**: Other. In our evaluation, we search the integral distinguisher on clock-reduced Atom. To find the integral distinguisher, we explore the propagation of the division property proposed by Todo [Tod15], which can search the integral distinguisher in more detail than the integral property, with an MILP-based search method proposed by Xiang et al. [XZBL16], which can efficiently explore the propagation of the bit-based division property. When we search the integral distinguisher, we give IV having **A**, which denotes that all bits in IV are active, as the division property at the input and then we check whether the output of the keystream bit is balanced after r clocks or not. As a result, we found the integral distinguisher after 67 clocks and we could not find the integral distinguisher after 68 clocks. Thus, we expect that Atom can resist against the integral attack. In addition, the division property was introduced to evaluate cube attacks [DS09], i.e. it evaluates the set of key bits J involved in the superpoly given a certain cube I [TIHM17]. Therefore, this result shows that the full rounds of Atom has a sufficient security level against cube attacks.

4.5 Algebraic/Guess-and-Determine Attacks

The output functions in Atom were chosen to be sufficiently complicated to prevent advances due to algebraic attacks. In [BGJ08], an algebraic attack was proposed against Grain-like ciphers in which the NFSR variables add only linearly to the expression for the keystream bit. For example it was shown that if a modified version of the Grain v1 cipher was conceived in such a way that it contained only the non-linear register from which the keystream was obtained only by linearly adding specific bits of the inner state, then each updated NFSR bit would be a linear function of initial state of NFSR bits. For example if $v_0, v_1, v_2, \ldots, v_{n-1}$ is the initial state of the NFSR such that each updated bit is calculated as $v_{n+t} = G(v_{n+t-1}, v_{n+t-2}, \ldots, v_t)$ and each keystream bit $z_t = L(v_{n+t-1}, v_{n+t-2}, \ldots, v_t)$, where $G,\ L$ are non-linear and linear boolean functions respectively on n bits, then updated bit v_{n+t} can be written as

$$v_{n+t} = G(v_{n+t-1}, v_{n+t-2}, \ldots, v_t) = \ Lin\ (v_0, v_1, \ldots, v_{n-1}, \ \ z_0, z_1, \ldots, z_t),\ \forall t \qquad (4)$$

Here Lin is another linear function. The above is not difficult to show and requires simple mathematical induction based arguments: if t_0 is the smallest integer for which the expression for z_{t_0} contains the term v_n, then from the linearity of z_{t_0}, we can see that v_n can be written as a linear expression in $v_0, v_1, \cdots, v_{n-1}$ and z_{t_0}. The argument carries forward in a similar manner for any subsequent value of $t = t_0 + 1, t_0 + 2, \ldots$ etc. Now we can multiply Equation (4) on both sides by H which is the annihilator of G to get equations of lower algebraic degree. The authors of [BGJ08] showed that for the modified of Grain v1, one could generate degree 4 equations using the annihilator of the non-linear update function. The system could then be linearized and solved using Gaussian elimination using 2^{49} operations.

Consider what happens when we have a Grain-like structure in which the attacker somehow gets to know the entire LFSR state. If the output equation for the keystream bit only contains terms from the NFSR which are linearly added along with non-linear terms from the LFSR, then due to the fact that the LFSR is completely known, the expression for the keystream bit becomes a purely linear expression in the initial NFSR state variables and we arrive at a situation that is similar to the one described in the previous paragraph. For Atom, we made sure that the expression for the kesytream bit contains higher degree terms with NFSR bits. Hence the attack of [BGJ08] does not apply to Atom.

In [BBI19, MAM16, Ban15] algebraic attacks via the method of SAT solvers are proposed against Sprout and Plantlet. The idea is to formulate equations relating the key

Table 2: Experimental data for algebraic attack. The Effective Complexity has been computed in terms of number of encryptions. Threshold runtime is the maximum time any equation solver can theoretically take for the total effective complexity to be below exhaustive search.

# NFSR bits	Runtime (secs)			Effective	Threshold
guessed	Average	Maximum	Minimum	Complexity	Runtime (secs)
65	398.509	1022.764	137.549	$2^{145.0}$	3.016
70	16.149	51.855	0.636	$2^{145.4}$	0.094
75	0.700	1.344	0.400	$2^{145.9}$	0.003
80	0.449	0.832	0.340	$2^{150.2}$	$9.204 \cdot 10^{-5}$

and the internal state variables to the keystream bits and forward the resulting equation bank to a suitable solver. This approach is in itself slightly problematic against Atom since a part of the LFSR directly decides which key bit is used to update the NFSR. If the LFSR is variable or unknown, the attacker would find it difficult to enumerate any equation bank as he wouldn't know which key bit was used in the state update. Hence to use this approach the attacker has to guess correctly the entire LFSR state in order to proceed with the attack (this imposes a multiplicative complexity 2^{60} to begin with, since the attacker knows that the last 9 bits of the LFSR at the beginning of the keystream phase is always 1). After this, we tried to follow a similar approach as in the above papers and present an equation bank to the solver. On a machine running with an Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz and 12 GB RAM, we could not solve the equations in reasonable time unless we additionally guessed correctly the values of NFSR bits too. In Table 2 we list the average runtimes taken by the solver to find a solution, against the number of NFSR bits guessed. The averages were calculated for 100 random instances of Atom keystream. The bits were guessed from the most significant end of the NFSR, so that they would remain in the NFSR longer during the evolution of the state and help simplify the algebraic complexity of the equations.

The solver failed to return any solution in reasonable time for less than 65 guesses. It can be seen that for 65 guesses of NFSR bits, we already have a complexity of $2^{60+65} = 2^{125}$ to account for the guesses. Finally, we estimated the amount of time needed to perform one Atom encryption. Since there is no straightforward way to compute the number of steps taken by the solver to solve a given polynomial system, there is no good way of comparing the computational costs of solving an equation and performing one encryption. Due to this fact, many papers [ZLFL14, MAM16, BBI19] in the past have measured the physical time to perform the above tasks to make a comparison. In [MAM16], in order to estimate the computational complexity of guess and determine attacks, the authors had measured the time of performing one encryption and concluded that it was possible to perform around 2^{10} encryptions per second on their system. Using this fact and after experimentally finding the average physical time required to solve a particular set of equations, they had concluded that guess and determine attacks on Plantlet did not perform better than a brute force attack. We adopt a similar method to estimate the bounds we present in this paper.

As argued in [EK15, Ban15, BBI19] one Atom encryption should be equal to the average number of rounds required to be executed per trial with a guessed value of the key (in a brute force search). This comes to 511 initialization rounds and 4 rounds in the keystream generation phase. We have given a proof of this in Appendix A (at the end of this paper). We calculated the average time required to execute one encryption (i.e. 511 initialization and 4 keystream rounds) on the same setup for 10000 randomly generated test vectors. We found that the average time required to compute one encryption is around 0.377 ms, which amounts to $E \approx 2652.5$ encryptions per second. The total complexity (in terms of number of encryptions) may be estimated as $2^{60+g} \cdot E \cdot T$, where T is roughly the average number of time taken to solve an equation, and g is the number of NFSR bits guessed. The results are tabulated in Table 2. In can be seen that Atom is secure against algebraic/guess and determine attacks.

For a hardware oriented cipher, exhaustive search should ideally be measured by the efficiency of a hardware implementation. However in most previous papers such as [ZLFL14, MAM16, BBI19], attacks are software oriented and the cost in encryptions has been calculated on the basis of the software attack. An exhaustive search in hardware can run as fast as the underlying technology allows it to. For example in Table 3 we can see that Atom can produce a maximum throughput of 490.5 Mbps on a circuit constructed with STM 90nm standard cells, 800 Mbps on Nangate 45 nm, and over 1 Gbps on TSMC 28 nm standard cells. This corresponds to $E = 6.06 \cdot 10^5,\ 1.55 \cdot 10^6,\ 2.09 \cdot 10^6$ encryptions per second on the 3 respective platforms. For example if exhaustive search were to be run on a custom made ASIC chip constructed with TSMC 28 nm transistors, using our circuit, we would be able to verify $2.09 \cdot 10^6$ keys per second. However to estimate the cost of algebraic attack fairly, we need to run our equation solver in hardware on the same chip. At the moment we do not have access to such a solver: while software based SAT solvers are easily available, we do not know if there exists such a custom circuit on hardware. Hence our benchmarking is done on software solvers.

4.6 Banik's Distinguishing Attack Against Sprout [Ban15]

Atom is secure against generic Time Memory Data (TMD) Tradeoff attacks as presented in [BS00], for the same reason that Sprout, Plantlet, Lizard are secure. The reason is that it is not possible to construct a one way function that maps the internal state to any keystream vector that does not additionally require the secret key. Furthermore the key update component in the state update function is completely linear, this ensures that table based special state attacks of [EK15] do not apply to all post-Sprout constructions. An interesting distinguishing attack against Sprout using slid keystreams was presented in [Ban15] that also applies to Plantlet, Lizard which was further generalized in [HKMZ18]. We will present the attack in context of Atom. Consider any random initial state $S_R \in \{0,1\}^{159}$. Since the state update function in both the keystream generation and key-IV initialization is bijective and efficiently invertible, we can apply both the Init^{-1} and Update^{-1} algorithms on it. Given the secret key, the former would reverse the entire key-IV initialization on any random string of 159 bits, and the latter inverts one round of the state update during keystream generation. A state S_R is a valid initial state after key-IV initialization, if a) its last 9 bits in decimal representation equals 511 and b) if $\mathsf{Init}^{-1}(S_R)$ has the 22 bit constant used to initialize Atom in bit positions 128 to 149. Thus the probability that a random S_R is a valid initial state is around $2^{-22-9} = 2^{-31}$. Similarly the probability that S_R is a valid t^{th} state after initialization is also 2^{-31} ($S = [\mathsf{Update}^{-1}]^t(S_R)$ and $\mathsf{Init}^{-1}(S)$ must satisfy the required conditions). Hence the probability that for any given key S_R is both the 0^{th} and t^{th} post-initialization state for 2 different IVs is around 2^{-62}. From randomness considerations we can therefore conclude that on average for every key there exists $2^{159-62} = 2^{97}$ IV pairs IV_1, IV_2 that satisfy such a condition. If t is such that the order and sequence of keybits that is used in the state update following the 0^{th} and t^{th} clocks are the same, then it is clear that the IV pair IV_1, IV_2 produce t-bit shifted keystream for the given key. So our distinguisher is as follows:

- Generate around $2t$ keystream bits $Z_1||Z_2$ for the unknown Key K and some randomly generated Initial Vector IV (where Z_1 and Z_2 are t-bit vectors each).
- Store the keystream bits in some appropriate data structure such as a hash table keyed with both Z_1 and Z_2 (to help easy detection of collisions),

- Continue the above steps with more randomly generated IVs IV till we obtain two Initial Vectors for K that generate t-bit shifted keystream.

Imagine the space of Initial Vectors as an undirected Graph $G = (W, E)$, where $W = \{0,1\}^{128}$ is the Vertex set which contains all the possible 128 bit Initial vector values as nodes. An edge $(IV_1, IV_2) \in E$ if and only if (K, IV_1) and (K, IV_2) produce t-bit shifted keystream sequence. From the above discussion, it is clear that the cardinality of E is expected to be 2^{97}. When we run the Distinguisher algorithm for N different Initial Vectors, we effectively add $\binom{N}{2}$ edges to the coverage and a match occurs when one of these edges is actually a member of the Edge-set E. Since there are potentially $\binom{2^{128}}{2}$ edges in the IV space, by the Birthday bound, a match will occur when the product of $\binom{N}{2}$ and the cardinality of E which is around 2^{97} is equal to $\binom{2^{128}}{2}$. From this equation solving for N, we get $N \approx 2^{79.5} = \sqrt{2^{159}}$ which is square root of the cardinality of the state space. This gives a bound for the time and memory complexity of the Distinguisher. The time complexity is around $\sqrt{2^{159}}$ encryptions, and the memory required is of the order of $2t \cdot \sqrt{2^{159}}$ bits.

This keystream distinguisher also works for Atom, but we claim that this can not be converted to a key recovery attack. Consider what happens when the attacker finds two IVs V_1, V_2 that produces 128-bit shifted keystream for some secret key K. This implies that there exists a state S_R which is the 0^{th} and 128^{th} post-initialization state after initialization with key-IV pairs (K, V_1) and (K, V_2) respectively. This implies the following two things

1. S_R and $[\mathsf{Update}^{-1}]^{128}(S_R)$ are such that the last 9 LFSR bits of both these states is the 9 bit string 1^9.
2. $\mathsf{Init}^{-1} S_R$ and $\mathsf{Init}^{-1} \circ [\mathsf{Update}^{-1}]^{128}(S_R)$ are such that the last 31 LFSR bits of both these states is the 31-bit constant used to initialize Atom.

Of these, the latter is not of much use cryptographically, since Init^{-1} is an algebraically complex function, most probably of degree close to $(128 + 159)$. However, Update^{-1} is a linear function on the LFSR part of the state. The statement "last 9 bits of S_R equals 1" implies that the S_R can be denoted as the symbolic variable string $\ell_0, \ell_1, \ldots, \ell_{59}, 1^9$ over GF(2). The statement "last 9 bits of $(\mathsf{Update}^{-1})^{128}(S_R)$ equals 1" is a set of 9 linear equations over the 60 variables ℓ_i. The kernel of this system has dimension 51, which implies that there are 2^{51} possible values that the LFSR part of S_R can have.

Hereafter, the attacker may use the equation solving approach used in the previous subsection to solve for the NFSR state and the key. The only difference is that the attacker now has fewer number of LFSR states to try out (2^{51} is this attack compared to 2^{60} in the pure guess and determine attack in the previous subsection). This implies that the total complexity required for this approach is faster than the attack complexities listed in Table 2 by a factor of 2^9, plus an additive complexity of $2^{79.5}$ required to find the shifted keystreams. This is still worse than exhaustive search and requires memory of $2t \cdot 2^{79.5} \approx 2^{87.5}$ bits.

5 Hardware Evaluation

We implemented all the ciphers using the standard cell libraries based on the STM 90nm, Nangate 45nm and TSMC 28 nm logic processes[1]. The following design flow was adhered to. All the designs were initially implemented in VHDL and the functional verification was done using *Mentor Graphics ModelSim SE* software. The designs were then synthesized using the *Synopsys Design Compiler* for the Standard Cell libraries of the three logic processes mentioned above. The switching activity file was then generated by performing a

[1]All source codes are publicly available in `https://github.com/qantik/atom`.

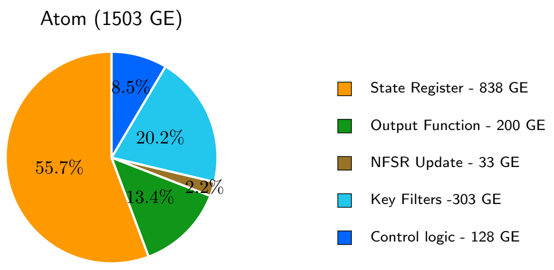

Figure 4: Breakdown of the area of individual components for the STM 90 nm process using the `compile_ultra -no_autoungroup` synthesis directive.

timing simulation on the synthesized netlist using the *Synopsys VCS* Software. The power was then estimated with the *Synopsys Power Compiler* by using the switching activity file. In Table 3, we compare our implementation results with current state of the art hardware stream ciphers providing 80-bit security Grain v1, Trivium, Sprout, Plantlet, Lizard and the 128-bit stream cipher Grain-128. For completeness, we also include AES-CTR in our benchmarks. There are numerous hardware implementations of AES presented in literature: from fully unrolled which performs encryption in one clock cycle, to round based which takes 10 clock cycles per encryption to various serialized circuits which although smaller in hardware size, utilize much more clock cycles for the same purpose. We choose the byte-serial AES implementation presented in [BCB20] which takes around 176 clock cycles to encrypt one plaintext and so when used in counter mode, this circuit can encrypt 128 bits every 176 clock cycles (this is better than the 216/246 cycle implementations of [BBR16, BBR19]). This is close to the 1 bit/cycle implementations of the other stream ciphers we have listed in Table 3.

As an instructive example, in Figure 4, we present a break-up of the area shares taken by the various components of the circuit in the design of Atom using the standard cell library of the STM 90nm logic process. As can be seen, a major part of the circuit area (around 56%) is occupied just by the registers. The two key filters occupy 303 GE in total, which means that the additional innovation cost us around 150 GE. As can be seen our design is quite competitive in performance as compared with Grain 128.

A high level diagram of the circuit is present in Figure 5. As can be seen Init represents the signal which is high only during the initialization phase. The LFSR is indeed partitioned in to two sections of 60 and 9 bits each. We also have a dedicated decimal counter that updates the last 9 bits of the LFSR during initialization which is discontinued as soon as the initialization phase is completed. The key filters are arranged so that the $k_{t\%128}$ component is added to the NFSR update only after the initialization phase is completed.

A note about throughput: If any application requires higher throughput relative to hardware area, Table 3 shows that this can be achieved with Grain or Trivium. Hence for some speeds, these ciphers offer a much better area/performance tradeoff. Also note that, for Grain/Trivium it is possible to compute a lot of bits in parallel relatively easily, because these designs deliberately leave last few bits in the registers untapped. While these designs can afford to do this, because their register sizes are 2-3 times the key size, for small-state designs this is undesirable, for following reasons: **(1)** It decreases the sampling resistance of the cipher and so a TMD-tradeoff attack via BSW sampling may become feasible, **(2)** prevent situations like [Ban15, Section 5.2] in which if part of the LFSR with some probability is zero for few clock cycles, the keystream can be expressed as a function of a much smaller part of the internal state, and **(3)** another reason why parallelizing

[2]We implemented AES in the counter mode of operation in a byte-serial way equipped with a 32-bit counter.

Table 3: Hardware measurements of Atom and other lightweight stream cipher constructions. For this comparison all listed algorithms have been implemented using three different cell library processes, i.e., STM 90 nm, NanGate 45 nm and TSMC 28 nm. All constructions were synthesized using the `compile_ultra` directive.

	State/Key Size	Area		Latency	Throughput	TP/Area	Power	Energy
		(μm^2)	(GE)	(ns)	(Mbit/s)	(10^6 Mbit/s·m^2)	(μW, 100 MHz)	(pJ/bit)
STM 90 nm								
Grain-v1 [HJM07]	160/80	4300.40	979.52	1.65	606.06	0.141	372.7	3.78
Trivium [Can06]	288/80	6597.57	1502.7	1.83	546.45	0.083	607.7	6.21
Sprout [AM15]	80/80	2990.96	681.25	1.87	534.76	0.179	234.8	2.37
Plantlet [MAM16]	101/80	3436.58	782.70	1.87	534.76	0.156	276.4	2.79
Lizard [HKM17]	121/120	5490.19	1250.5	2.58	387.63	0.071	358.2	3.65
Grain-128 [HJMM06]	256/128	6397.91	1457.2	2.22	450.45	0.070	562.9	5.75
Kreyvium [CCF+18]	288/128	12970.3	2954.2	2.04	490.21	0.038	1019	10.60
Trivium-MB [MB07]	288/128	7085.00	1613.7	2.06	485.44	0.069	614.5	6.29
Trivia [CCHN15]	384/128	8964.09	2041.7	2.06	485.44	0.054	803.5	8.29
AES-CTR [2]	288/128	9191.35	2093.5	6.16	118.06	0.013	1099.0	15.10
Atom	159/128	6543.89	1490.5	3.20	312.51	0.047	490.5	5.02
NanGate 45 nm								
Grain-v1	160/80	1161.93	1456.02	0.63	1587.38	1.367	342.5	3.50
Trivium	288/80	1857.58	2327.79	0.51	1960.78	1.056	548.3	5.68
Sprout	80/80	768.740	963.331	0.62	1612.90	2.098	227.0	2.30
Plantlet	101/80	873.277	1094.32	0.76	1315.79	1.507	242.0	2.46
Lizard	121/120	1326.00	1661.65	0.81	1234.57	0.931	386.9	3.97
Grain-128	256/128	1685.64	2112.33	0.87	1149.43	0.682	453.5	4.68
Kreyvium	288/128	3441.55	4312.66	0.76	1315.79	0.382	824.9	8.80
Trivium-MB	288/128	1947.91	2440.99	0.51	1960.78	1.007	128.8	1.33
Trivia	384/128	2482.58	3111.00	0.73	1369.86	0.552	725.1	7.60
AES-CTR	288/128	2340.28	2932.68	2.05	354.767	0.152	729.5	10.00
Atom	159/128	1613.29	2021.67	1.25	800.000	0.496	431.8	4.45
TSMC 28 nm								
Grain-v1	160/80	493.53	1353.99	0.81	1234.57	2.502	101	1.04
Trivium	288/80	730.74	2004.66	0.61	1639.34	2.243	163	1.68
Sprout	80/80	333.27	914.325	0.50	2000.00	6.001	60.8	0.62
Plantlet	101/80	384.18	1053.99	0.54	1851.85	4.820	72.1	0.73
Lizard	121/120	596.93	1637.67	0.92	1086.96	1.821	86.9	0.89
Grain-128	256/128	736.77	2021.32	0.57	1754.39	2.381	155	1.60
Kreyvium	288/128	1455.6	3993.33	0.62	1612.90	1.108	284	3.02
Trivium-MB	288/128	806.03	2211.33	0.65	1538.46	1.909	168	1.74
Trivia	384/128	1004.8	2756.65	0.66	1515.15	1.508	200	2.08
AES-CTR	288/128	1054.3	2892.35	1.62	448.93	0.526	254	3.49
Atom	159/128	720.61	1976.98	0.93	1075.27	1.492	121	1.25

is counterproductive in Atom/ciphers with key-filter, is that for any x-times parallelized implementation, we will need $2x$ keybits to update the state, which requires $2x$ key-filters. Since each filter is a 128-to-1 multiplexer, this increases the area of the design multiple folds.

6 Conclusion

In this paper we present the stream cipher Atom with internal state only around 25% larger than the secret key. Since all such attempts in the past had some cryptanalytic advances reported against them, our aim was to see if there were any high level architectural modifications that could make such designs immune against common cryptanalytic methods. As a result we adopted a Grain-like structure with an additional key filter that seems to protect against most cryptanalytic advances reported against small state ciphers. We performed an extensive review of the design with detailed hardware implementations in 2 standard cell libraries to support our findings.

Acknowledgments

The authors would like to thank Yann Rotella and the anonymous ToSC reviewers for their valuable comments and suggestions. Subhadeep Banik is supported by the Swiss National Science Foundation (SNSF) through the Ambizione Grant `PZ00P2_179921`. Takanori Isobe

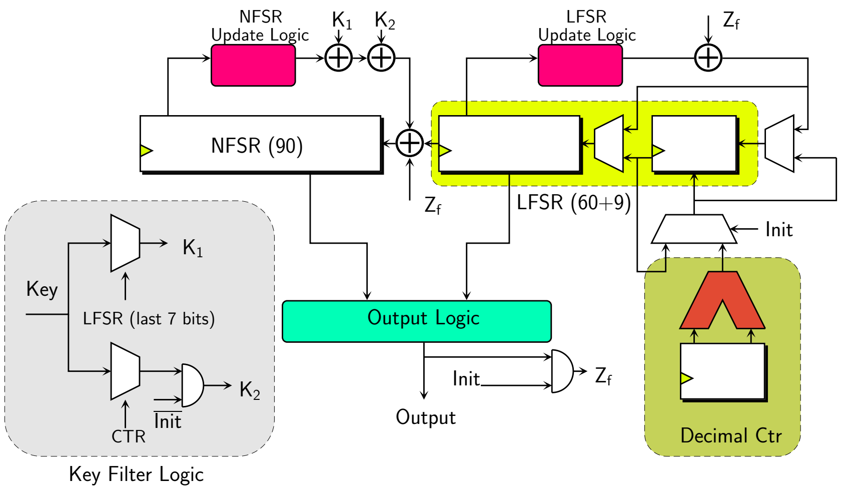

Figure 5: High level Circuit Diagram. Note that the logic required to load the IV and constant pad on to the registers is not explicitly shown.

is supported by JST, PRESTO Grant Number JPMJPR2031, Grant-in-Aid for Scientific Research (B)(KAKENHI 19H02141) and JSPS bilateral Program with DST (JPJSBP 120197735). Kosei Sakamoto is supported by Grant-in-Aid for JSPS Fellows (KAKENHI 20J23526) for Japan Society for the Promotion of Science.

References

[AM15] Frederik Armknecht and Vasily Mikhalev. On Lightweight Stream Ciphers with Shorter Internal States. In *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, pages 451–470, 2015.

[Bab95] S. H. Babbage. Improved “exhaustive search” attacks on stream ciphers. In *European Convention on Security and Detection, 1995.*, pages 161–166, 1995.

[Ban15] Subhadeep Banik. Some Results on Sprout. In *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings*, pages 124–139, 2015.

[Ban16] Subhadeep Banik. Conditional differential cryptanalysis of 105 round grain v1. *Cryptogr. Commun.*, 8(1):113–137, 2016.

[BB93] Ishai Ben-Aroya and Eli Biham. Differential Cryptanalysis of Lucifer. In *Advances in Cryptology - CRYPTO ’93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, pages 187–199, 1993.

[BBI19] Subhadeep Banik, Khashayar Barooti, and Takanori Isobe. Cryptanalysis of Plantlet. *IACR Trans. Symmetric Cryptol.*, 2019(3):103–120, 2019.

[BBR16] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-aes: A compact implementation of the AES encryption/decryption core. In *Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings*, pages 173–190, 2016.

[BBR19] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Compact circuits for combined AES encryption/decryption. *J. Cryptogr. Eng.*, 9(1):69–83, 2019.

[BCB20] Fatih Balli, Andrea Caforio, and Subhadeep Banik. The Area-Latency Symbiosis: Towards Improved Serial Encryption Circuits. *IACR Cryptol. ePrint Arch.*, 2020:608, 2020.

[BGJ08] Côme Berbain, Henri Gilbert, and Antoine Joux. Algebraic and Correlation Attacks against Linearly Filtered Non Linear Feedback Shift Registers. In *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, pages 184–198, 2008.

[BGM06] Côme Berbain, Henri Gilbert, and Alexander Maximov. Cryptanalysis of Grain. In *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, pages 15–29, 2006.

[BICG17] Subhadeep Banik, Takanori Isobe, Tingting Cui, and Jian Guo. Some cryptanalytic results on Lizard. *IACR Trans. Symmetric Cryptol.*, 2017(4):82–98, 2017.

[Bjo08] Tor E. Bjorstad. Cryptanalysis of Grain using Time/Memory/Date Tradeoffs. eSTREAM, ECRYPT Stream Cipher Project, Report 2008/012, 2008. https://www.ecrypt.eu.org/stream/papersdir/2008/012.pdf.

[BS90] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, pages 2–21, 1990.

[BS91] Eli Biham and Adi Shamir. Differential Cryptanalysis of Feal and N-Hash. In *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, pages 1–16, 1991.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, pages 1–13, 2000.

[Can06] Christophe De Cannière. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In Sokratis K. Katsikas, Javier López, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

[Can11] Anne Canteaut. A5/1. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security*, pages 1–2. Springer US, Boston, MA, 2011.

[CCF$^+$18] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. *J. Cryptol.*, 31(3):885–916, 2018.

[CCHN15] Avik Chakraborti, Anupam Chattopadhyay, Muhammad Hassan, and Mridul Nandi. TriviA: A Fast and Secure Authenticated Encryption Scheme. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 330–353. Springer, 2015.

[DK08] Orr Dunkelman and Nathan Keller. Treatment of the initial value in Time-Memory-Data Tradeoff attacks on stream ciphers. *Inf. Process. Lett.*, 107(5):133–137, 2008.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square. In *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, pages 149–165, 1997.

[DS09] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 278–299, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[EK15] Muhammed F. Esgin and Orhun Kara. Practical Cryptanalysis of Full Sprout with TMD Tradeoff Attacks. In *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, pages 67–85, 2015.

[Fon11] Caroline Fontaine. RC4. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security*, pages 1031–1032. Springer US, Boston, MA, 2011.

[Gol97] Jovan Dj. Golic. Cryptanalysis of Alleged A5 Stream Cipher. In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, pages 239–255, 1997.

[Hel80] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980.

[HJM07] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *Int. J. Wirel. Mob. Comput.*, 2(1):86–93, 2007.

[HJMM06] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A Stream Cipher Proposal: Grain-128. In *Proceedings 2006 IEEE International Symposium on Information Theory, ISIT 2006, The Westin Seattle, Seattle, Washington, USA, July 9-14, 2006*, pages 1614–1618. IEEE, 2006.

[HK05] Jin Hong and Woo-Hwan Kim. TMD-Tradeoff and State Entropy Loss Considerations of Streamcipher MICKEY. In Subhamoy Maitra, C. E. Veni Madhavan, and Ramarathnam Venkatesan, editors, *Progress in Cryptology - INDOCRYPT 2005*, pages 169–182, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[HKM17] Matthias Hamann, Matthias Krause, and Willi Meier. LIZARD - A Lightweight Stream Cipher for Power-constrained Devices. *IACR Trans. Symmetric Cryptol.*, 2017(1):45–79, 2017.

[HKMZ18] Matthias Hamann, Matthias Krause, Willi Meier, and Bin Zhang. Design and analysis of small-state Grain-like stream ciphers. *Cryptogr. Commun.*, 10(5):803–834, 2018.

[HS05] Jin Hong and Palash Sarkar. New Applications of Time Memory Data Tradeoffs. In *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, pages 353–372, 2005.

[Inc15] Gurobi Optimization Inc. Gurobi Optimizer 6.5. Official webpage, http://www.gurobi.com/, 2015.

[KMN10] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2010.

[KMN11] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional Differential Cryptanalysis of Trivium and KATAN. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 200–212. Springer, 2011.

[KW02] Lars R. Knudsen and David A. Wagner. Integral Cryptanalysis. In *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers*, pages 112–127, 2002.

[KY10] Selçuk Kavut and Melek Diker Yücel. 9-variable Boolean functions with nonlinearity 242 in the generalized rotation symmetric class. *Inf. Comput.*, 208(4):341–350, 2010.

[LN15] Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of Full Sprout. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 663–682, 2015.

[Luc01] Stefan Lucks. The Saturation Attack - A Bait for Twofish. In *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, pages 1–15, 2001.

[MAM16] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On Ciphers that Continuously Access the Non-Volatile Key. *IACR Trans. Symmetric Cryptol.*, 2016(2):52–79, 2016.

[MB07] Alexander Maximov and Alex Biryukov. Two Trivial Attacks on Trivium. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, volume 4876 of *Lecture Notes in Computer Science*, pages 36–55. Springer, 2007.

[MSS$^+$18] Subhamoy Maitra, Nishant Sinha, Akhilesh Siddhanti, Ravi Anand, and Sugata Gangopadhyay. A TMDTO Attack Against Lizard. *IEEE Trans. Computers*, 67(5):733–739, 2018.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In *Information Security and Cryptology - 7th International Conference, Inscrypt 2011,* *Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *LNCS*, pages 57–76, 2011.

[TIHM17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube Attacks on Non-Blackbox Polynomials Based on Division Property. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, pages 250–279, 2017.

[TIM$^+$18] Yosuke Todo, Takanori Isobe, Willi Meier, Kazumaro Aoki, and Bin Zhang. Fast Correlation Attack Revisited - Cryptanalysis on Full Grain-128a, Grain-128, and Grain-v1. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 129–159, 2018.

[TMA19] Yosuke Todo, Willi Meier, and Kazumaro Aoki. On the Data Limitation of Small-State Stream Ciphers: Correlation Attacks on Fruit-80 and Plantlet. In *Selected Areas in Cryptography - SAC 2019 - 26th International Conference, Waterloo, ON, Canada, August 12-16, 2019, Revised Selected Papers*, pages 365–392, 2019.

[Tod15] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 287–314, 2015.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 648–678, 2016.

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

[ZLFL14] Bin Zhang, Zhenqi Li, Dengguo Feng, and Dongdai Lin. Near Collision Attack on the Grain v1 Stream Cipher. In Shiho Moriai, editor, *Fast Software Encryption*, pages 518–538, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[ZXM18] Bin Zhang, Chao Xu, and Willi Meier. Fast Near Collision Attack on the Grain v1 Stream Cipher. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 771–802, 2018.

A Number of rounds equaling one Atom encryption [EK15, BBI19]

To do an exhaustive search, first an initialization phase has to be run for 511 rounds, and then generate 128 -bits of keystream to do a unique match. However, since each keystream bit generated matches the correct one with probability $\frac{1}{2}$, 2^{128} keys are tried for

1 clock and roughly half of them are eliminated, 2^{127} for 2 clocks and half of the remaining keys are eliminated, and so on. This means that in the process of brute force search, the probability that for any random key, $(i+1)$ Atom keystream phase rounds need to be run, is $\frac{1}{2^i}$. Hence, the expected number of Atom keystream rounds per trial is

$$\sum_{i=0}^{127} \frac{(i+1)2^{128-i}}{2^{128}} = \sum_{i=0}^{127} (i+1)\frac{1}{2^i} \approx 4$$

B Test vectors

1. $Key =$ `0000 0000 0000 0000 0000 0000 0000 0000`
   $IV =$ `0000 0000 0000 0000 0000 0000 0000 0000`
   $Z =$ `8ddb 7baf 22c4 ce3a b3bc 350f aa13 552b`

2. $Key =$ `aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa`
   $IV =$ `5555 5555 5555 5555 5555 5555 5555 5555`
   $Z =$ `0906 c183 66a8 0c65 8f8b 2c8b 6c61 f978`

3. $Key =$ `bcd6 0b1e 3af4 a91d 5d52 8342 18e8 9d7f`
   $IV =$ `0000 0000 0000 0000 0000 0000 0000 0000`
   $Z =$ `ba1c 3e51 eaad 415a ca7b 41f2 cf79 a017`

4. $Key =$ `e933 7ce2 2191 2ea7 e481 b3d9 a630 564e`
   $IV =$ `6e16 4e17 fc18 487f bbbf 01af b42f 4545`
   $Z =$ `3d16 c749 2881 d2c5 8547 0bd0 e1ef 50f1`

5. $Key =$ `cd71 8f56 a392 6747 6758 7feb 4906 a5d4`
   $IV =$ `ba3c 55a8 d8e2 7e97 7534 b17d 7bf5 40c5`
   $Z =$ `d0d2 370e e7d6 9587 67c5 56cf 7b85 b11e`

6. $Key =$ `e90e 92e6 fc7c caeb ff72 3369 1cc2 0350`
   $IV =$ `8118 9ff9 ca94 999c faaa a700 e9b3 a82c`
   $Z =$ `8d41 b026 3e6b cf60 56b7 f3a3 54fa c80c`

C Search for Linear Masks

Let us give one example of the search that we undertook to mount a correlation attack. First define $\theta_{90+t} = b_{89}^{t+1} \oplus G(B^t) \oplus l_0^t \oplus k_{cnt} \oplus k_{t\%128}$ which is identically equal to 0. As per the terminology introduced in [TMA19], we try to find linear masks $\mathbb{T}_Z$ and $\mathbb{T}_\theta$ such that the resulting expression for the sum of keystream bits is the simplest and likely to yield high correlation between sum of keystream, keybits and LFSR bits. After exhaustively looking through the search space we found $\mathbb{T}_Z = \{1, 3, 8\}$ and $\mathbb{T}_\theta = \{1, 4, 5, 7, 8\}$. Now we have

$$z^t \oplus z^{t+3} \oplus z^{t+8} = z^t \oplus z^{t+3} \oplus z^{t+8} \oplus \theta_{91+t} \oplus \theta_{94+t} + \oplus \theta_{95+t} \oplus \theta_{97+t} \oplus \theta_{98+t} = F(B^t, L^t) \oplus G(K),$$

where F is a function of the NFSR state B^t and LFSR state L^t and G is a function of key bits K. The function $F(B^t, L^t)$ contains 428 monomials over 107 variables. We write F as a sum of four functions F_1, F_2, F_3 and F_4 such that number of variables of each function is less than 37.

$$\begin{aligned}
F_1 &= F_1(b^t_4, b^t_7, b^t_8, b^t_{10}, b^t_{11}, b^t_{13}, b^t_{32}, b^t_{33}, b^t_{56}, b^t_{59}, b^t_{60}, b^t_{61}, b^t_{62}, b^t_{63}, b^t_{64}, b^t_{65},\\
&\quad b^t_{66}, b^t_{67}, b^t_{68}, b^t_{69}, b^t_{70}, b^t_{72}, b^t_{73}, b^t_{75}, b^t_{76}, b^t_{77}, b^t_{78}, b^t_{79}, b^t_{80}, b^t_{81}, b^t_{82}, b^t_{84}),\\
F_2 &= l^t_4 \oplus l^t_8 l^t_{19} \oplus l^t_8 \oplus l^t_{21} l^t_{23} \oplus l^t_{30} l^t_{42} \oplus l^t_{30} l^t_{75} \oplus b^t_{18} b^t_{20} \oplus b^t_{19} b^t_{20} \oplus b^t_{19} \oplus b^t_{20} b^t_{21}\\
&\quad \oplus b^t_{21} b^t_{23} b^t_{24} \oplus b^t_{22} b^t_{23} \oplus b^t_{22} \oplus b^t_{23} b^t_{24} \oplus b^t_{24} b^t_{26} b^t_{27} \oplus b^t_{25} b^t_{27} b^t_{28} \oplus b^t_{26} b^t_{54}\\
&\quad \oplus b^t_{27} b^t_{29} b^t_{30} \oplus b^t_{28} b^t_{30} b^t_{31} \oplus b^t_{28} \oplus b^t_{29} b^t_{57} \oplus b^t_{29} \oplus b^t_{30} b^t_{58} \oplus b^t_{30} \oplus b^t_{31}\\
&\quad \oplus b^t_{54} \oplus b^t_{57},\\
F_3 &= l^t_1 \oplus l^t_5 l^t_{16} \oplus l^t_5 \oplus l^t_{16} l^t_{18} \oplus b^t_{36} b^t_{43} \oplus b^t_{36} \oplus b^t_{39} b^t_{46} \oplus b^t_{39} \oplus b^t_{40} b^t_{47} \oplus b^t_{43} b^t_{50}\\
&\quad \oplus b^t_{50} \oplus b^t_{86} \oplus b^t_{17} b^t_{19} \oplus b^t_{81} b^t_{84} b^t_{85} b^t_{87} \oplus b^t_{82} b^t_{85} b^t_{86} b^t_{88} \oplus b^t_{84} b^t_{87} b^t_{88} b^{t+1}_{89},\\
F_4 &= F_4(l^t_7, l^t_{10}, l^t_{13}, l^t_{15}, l^t_{22}, l^t_{24}, l^t_{25}, l^t_{33}, l^t_{36}, l^t_{38}, l^t_{41}, l^t_{45}, l^t_{46}, l^t_{50}, l^t_{53}, l^t_{58}, l^t_{59}, l^t_{62}, l^t_{65},\\
&\quad l^t_{67}, l^t_{70}, b^t_9, b^t_{12}, b^t_{14}, b^t_{15}, b^t_{16}, b^t_{17}, b^t_{41}, b^t_{42}, b^t_{44}, b^t_{49}, b^t_{85}, b^t_{87}, b^t_{88}, b^t_{89},\\
&\quad b^{t+2}_{89}, b^{t+4}_{89})
\end{aligned}$$

The expressions for F_1, F_4 are omitted for space constraints. However the function F is completely balanced no matter how the values of the involved LFSR bits are guessed in the expression. Since b^t_{19} is the only common variable in the expressions F_2, F_3, to verify that F is balanced we checked that $Pr(F_2 = 0|b^t_{19} = 0) = Pr(F_2 = 0|b^t_{19} = 1) = 0.5$ and $Pr(F_3 = 0|b^t_{19} = 0) = Pr(F_3 = 0|b^t_{19} = 1) = 0.5$. There may exist, but it is highly unlikely, that there exists other linear masks which lead to correlation attack in time faster than exhaustive search.

D Conditional Differential Polynomials

For the sake of brevity, we denote by $x_0, x_1, \ldots, x_{127}$ the IV bits, i.e., $x_0, x_1, \ldots, x_{127} \doteq IV_0, IV_1, \ldots, IV_{127}$.

$$\begin{aligned}
f_1 = {} & x_1x_{42}x_{43}x_{44}x_{45} + x_1x_{42}x_{43}x_{44} + x_1x_{42}x_{43} + x_1x_{42}x_{44}x_{45}x_{99}\\
& + x_1x_{42}x_{44}x_{99} + x_1x_{42}x_{45} + x_1x_{42}x_{99} + x_1x_{42} + x_1x_{43}x_{44}x_{45}x_{98}\\
& + x_1x_{43}x_{44}x_{98} + x_1x_{43}x_{98} + x_1x_{44}x_{45}x_{98}x_{99} + x_1x_{44}x_{45}\\
& + x_1x_{44}x_{98}x_{99} + x_1x_{44} + x_1x_{45}x_{98} + x_1x_{98}x_{99} + x_1x_{98}\\
& + x_2x_{43}x_{44}x_{45} + x_2x_{43}x_{44} + x_2x_{43} + x_2x_{44}x_{45}x_{99}\\
& + x_2x_{44}x_{99} + x_2x_{45} + x_2x_{99} + x_2 + x_3x_{44}x_{45} + x_3x_{44} + x_3 + x_4x_{45}\\
& + x_4 + x_5x_{42}x_{43}x_{44}x_{45} + x_5x_{42}x_{43}x_{44} + x_5x_{42}x_{43}\\
& + x_5x_{42}x_{44}x_{45}x_{99} + x_5x_{42}x_{44}x_{99} + x_5x_{42}x_{45} + x_5x_{42}x_{99}\\
& + x_5x_{42} + x_5x_{43}x_{44}x_{45}x_{98} + x_5x_{43}x_{44}x_{98} + x_5x_{43}x_{98}\\
& + x_5x_{44}x_{45}x_{98}x_{99} + x_5x_{44}x_{45} + x_5x_{44}x_{98}x_{99} + x_5x_{44}\\
& + x_5x_{45}x_{98} + x_5x_{98}x_{99} + x_5x_{98} + x_5 + x_6x_{43}x_{44}x_{45}\\
& + x_6x_{43}x_{44} + x_6x_{43} + x_6x_{44}x_{45}x_{99} + x_6x_{44}x_{99} + x_6x_{45}\\
& + x_6x_{99} + x_7x_{44}x_{45} + x_7x_{44} + x_7 + x_8x_{45} + x_8\\
& + x_9x_{41}x_{42}x_{43}x_{44}x_{45}x_{97} + x_9x_{41}x_{42}x_{43}x_{44}x_{97} + x_9x_{41}x_{42}x_{43}x_{97}\\
& + x_9x_{41}x_{42}x_{44}x_{45}x_{97}x_{99} + x_9x_{41}x_{42}x_{44}x_{97}x_{99} + x_9x_{41}x_{42}x_{45}x_{97}\\
& + x_9x_{41}x_{42}x_{97}x_{99} + x_9x_{41}x_{42}x_{97} + x_9x_{41}x_{43}x_{44}x_{45}x_{97}x_{98}
\end{aligned}$$

$$\begin{aligned}
&+ x_9x_{41}x_{43}x_{44}x_{97}x_{98} + x_9x_{41}x_{43}x_{97}x_{98} + x_9x_{41}x_{44}x_{45}x_{97}x_{98}x_{99}\\
&+ x_9x_{41}x_{44}x_{45}x_{97} + x_9x_{41}x_{44}x_{97}x_{98}x_{99} + x_9x_{41}x_{44}x_{97} + x_9x_{41}x_{45}x_{97}x_{98}\\
&+ x_9x_{41}x_{97}x_{98}x_{99} + x_9x_{41}x_{97}x_{98} + x_9 + x_{40}\\
&+ x_{41}x_{42}x_{43}x_{44}x_{45}x_{97} + x_{41}x_{42}x_{43}x_{44}x_{45} + x_{41}x_{42}x_{43}x_{44}x_{97} + x_{41}x_{42}x_{43}x_{44}\\
&+ x_{41}x_{42}x_{43}x_{97} + x_{41}x_{42}x_{43} + x_{41}x_{42}x_{44}x_{45}x_{97}x_{99} + x_{41}x_{42}x_{44}x_{45}x_{99}\\
&+ x_{41}x_{42}x_{44}x_{97}x_{99} + x_{41}x_{42}x_{44}x_{99} + x_{41}x_{42}x_{45}x_{97}\\
&+ x_{41}x_{42}x_{45} + x_{41}x_{42}x_{97}x_{99} + x_{41}x_{42}x_{97} + x_{41}x_{42}x_{99}\\
&+ x_{41}x_{42} + x_{41}x_{43}x_{44}x_{45}x_{97}x_{98} + x_{41}x_{43}x_{44}x_{45}x_{98}\\
&+ x_{41}x_{43}x_{44}x_{97}x_{98} + x_{41}x_{43}x_{44}x_{98} + x_{41}x_{43}x_{97}x_{98}\\
&+ x_{41}x_{43}x_{98} + x_{41}x_{44}x_{45}x_{97}x_{98}x_{99} + x_{41}x_{44}x_{45}x_{97}\\
&+ x_{41}x_{44}x_{45}x_{98}x_{99} + x_{41}x_{44}x_{45} + x_{41}x_{44}x_{97}x_{98}x_{99}\\
&+ x_{41}x_{44}x_{97} + x_{41}x_{44}x_{98}x_{99} + x_{41}x_{44} + x_{41}x_{45}x_{97}x_{98} + x_{41}x_{45}x_{98}\\
&+ x_{41}x_{97}x_{98}x_{99} + x_{41}x_{97}x_{98} + x_{41}x_{98}x_{99} + x_{41}x_{98}\\
&+ x_{41} + x_{42}x_{43}x_{44}x_{45}x_{90} + x_{42}x_{43}x_{44}x_{45}x_{95}\\
&+ x_{42}x_{43}x_{44}x_{45}x_{97} + x_{42}x_{43}x_{44}x_{45}x_{98} + x_{42}x_{43}x_{44}x_{45} + x_{42}x_{43}x_{44}x_{90}\\
&+ x_{42}x_{43}x_{44}x_{95} + x_{42}x_{43}x_{44}x_{97} + x_{42}x_{43}x_{44}x_{98}\\
&+ x_{42}x_{43}x_{44} + x_{42}x_{43}x_{90} + x_{42}x_{43}x_{95} + x_{42}x_{43}x_{97}\\
&+ x_{42}x_{43}x_{98} + x_{42}x_{43} + x_{42}x_{44}x_{45}x_{90}x_{99}x_{42}x_{44}x_{45}x_{95}x_{99} + x_{42}x_{44}x_{45}x_{97}x_{99}\\
&+ x_{42}x_{44}x_{45}x_{98}x_{99} + x_{42}x_{44}x_{45}x_{99} + x_{42}x_{44}x_{90}x_{99}\\
&+ x_{42}x_{44}x_{95}x_{99} + x_{42}x_{44}x_{97}x_{99} + x_{42}x_{44}x_{98}x_{99} + x_{42}x_{44}x_{99}\\
&+ x_{42}x_{45}x_{90} + x_{42}x_{45}x_{95} + x_{42}x_{45}x_{97} + x_{42}x_{45}x_{98}\\
&+ x_{42}x_{45} + x_{42}x_{90}x_{99} + x_{42}x_{90} + x_{42}x_{95}x_{99} + x_{42}x_{95}\\
&+ x_{42}x_{97}x_{99} + x_{42}x_{97} + x_{42}x_{98}x_{99} + x_{42}x_{98} + x_{42}x_{99} + x_{42}\\
&+ x_{43}x_{44}x_{45}x_{90}x_{98} + x_{43}x_{44}x_{45}x_{91} + x_{43}x_{44}x_{45}x_{95}x_{98}\\
&+ x_{43}x_{44}x_{45}x_{96} + x_{43}x_{44}x_{45}x_{97}x_{98} + x_{43}x_{44}x_{45}x_{98}\\
&+ x_{43}x_{44}x_{45}x_{99} + x_{43}x_{44}x_{90}x_{98} + x_{43}x_{44}x_{91}\\
&+ x_{43}x_{44}x_{95}x_{98} + x_{43}x_{44}x_{96} + x_{43}x_{44}x_{97}x_{98} + x_{43}x_{44}x_{98}\\
&+ x_{43}x_{44}x_{99} + x_{43}x_{90}x_{98} + x_{43}x_{91} + x_{43}x_{95}x_{98} + x_{43}x_{96}\\
&+ x_{43}x_{97}x_{98} + x_{43}x_{98} + x_{43}x_{99} + x_{44}x_{45}x_{90}x_{98}x_{99} + x_{44}x_{45}x_{90}\\
&+ x_{44}x_{45}x_{91}x_{99} + x_{44}x_{45}x_{92} + x_{44}x_{45}x_{95}x_{98}x_{99}\\
&+ x_{44}x_{45}x_{95} + x_{44}x_{45}x_{96}x_{99} + x_{44}x_{45}x_{97}x_{98}x_{99} + x_{44}x_{45}x_{98}x_{99}\\
&+ x_{44}x_{90}x_{98}x_{99} + x_{44}x_{90} + x_{44}x_{91}x_{99} + x_{44}x_{92} + x_{44}x_{95}x_{98}x_{99}\\
&+ x_{44}x_{95} + x_{44}x_{96}x_{99} + x_{44}x_{97}x_{98}x_{99} + x_{44}x_{98}x_{99} + x_{45}x_{90}x_{98}\\
&+ x_{45}x_{91} + x_{45}x_{93} + x_{45}x_{95}x_{98} + x_{45}x_{96} + x_{45}x_{97}x_{98} + x_{45}\\
&+ x_{90}x_{98}x_{99} + x_{90}x_{98} + x_{91}x_{99} + x_{91} + x_{92} + x_{93} + x_{94} + x_{95}x_{98}x_{99}\\
&+ x_{95}x_{98} + x_{96}x_{99} + x_{96} + x_{97}x_{98}x_{99} + x_{97}x_{98} + x_{97} + x_{98}x_{99} + x_{99}.
\end{aligned}$$

$$\begin{aligned}
f_2 = {} & x_1x_{42}x_{43}x_{44} + x_1x_{42}x_{44}x_{99} + x_1x_{43}x_{44}x_{98} + x_1x_{44}x_{98}x_{99} + x_1x_{44}\\
&+ x_1 + x_2x_{43}x_{44} + x_2x_{44}x_{99} + x_2 + x_3x_{44} + x_5x_{42}x_{43}x_{44} + x_5x_{42}x_{44}x_{99}\\
&+ x_5x_{43}x_{44}x_{98} + x_5x_{44}x_{98}x_{99} + x_5x_{44} + x_5 + x_6x_{43}x_{44} + x_6x_{44}x_{99} + x_7x_{44} + x_7\\
&+ x_9x_{41}x_{42}x_{43}x_{44}x_{97} + x_9x_{41}x_{42}x_{44}x_{97}x_{99} + x_9x_{41}x_{43}x_{44}x_{97}x_{98}\\
&+ x_9x_{41}x_{44}x_{97}x_{98}x_{99} + x_9x_{41}x_{44}x_{97} + x_9 + x_{41}x_{42}x_{43}x_{44}x_{97} + x_{41}x_{42}x_{43}x_{44}\\
&+ x_{41}x_{42}x_{44}x_{97}x_{99} + x_{41}x_{42}x_{44}x_{99} + x_{41}x_{43}x_{44}x_{97}x_{98} + x_{41}x_{43}x_{44}x_{98}
\end{aligned}$$

$$
\begin{aligned}
&+ x_{41}x_{44}x_{97}x_{98}x_{99} + x_{41}x_{44}x_{97} + x_{41}x_{44}x_{98}x_{99} + x_{41}x_{44} + x_{42}x_{43}x_{44}x_{90} \\
&+ x_{42}x_{43}x_{44}x_{95} + x_{42}x_{43}x_{44}x_{97} + x_{42}x_{43}x_{44}x_{98} + x_{42}x_{43}x_{44} + x_{42}x_{44}x_{90}x_{99} \\
&+ x_{42}x_{44}x_{95}x_{99} + x_{42}x_{44}x_{97}x_{99} + x_{42}x_{44}x_{98}x_{99} + x_{42}x_{44}x_{99} + x_{42} \\
&+ x_{43}x_{44}x_{90}x_{98} + x_{43}x_{44}x_{91} + x_{43}x_{44}x_{95}x_{98} + x_{43}x_{44}x_{96} \\
&+ x_{43}x_{44}x_{97}x_{98} + x_{43}x_{44}x_{98} + x_{43}x_{44}x_{99} + x_{44}x_{90}x_{98}x_{99} + x_{44}x_{90} + x_{44}x_{91}x_{99} \\
&+ x_{44}x_{92} + x_{44}x_{95}x_{98}x_{99} + x_{44}x_{95} + x_{44}x_{96}x_{99} + x_{44}x_{97}x_{98}x_{99} + x_{44}x_{98}x_{99} \\
&+ x_{44} + x_{45} + x_{93} + x_{97}.
\end{aligned}
$$