Cryptanalysis of full round Fruit

Sabyasachi Dey and Santanu Sarkar

Indian Institute of Technology Madras,
Chennai 600 036, India.
E-mail: sabya.ndp@gmail.com, sarkar.santanu.bir@gmail.com

Abstract. In FSE 2015, Armknetcht et al. proposed a new technique to design stream cipher. This technique involves repeated use of keybits in each round of keystream bit generation. This idea showed the possibility to design stream ciphers where internal state size is significantly lower than twice the key size. They proposed a new cipher based on this idea, named Sprout. But soon Sprout was proved to be insecure. In Crypto 2015, Lallemand et al. proposed an attack on Sprout, which was 2^{10} times faster than the exhaustive search. But the new idea used in Sprout showed a new direction in the design of stream cipher, which led to the proposal of several new ciphers with small size of internal state.
Fruit is another cipher in this direction proposed recently where both the key size and state size are 80. So far, there is no attack against this cipher. In this paper, we attack full round Fruit by a divide-and-conquer method. We use several types of sieving to reduce the possible candidates for an internal state. Our attack is equivalent to $2^{74.95}$ many Fruit encryption, which is around 16.95 times faster than average exhaustive key search. This is the first proposed attack against Fruit.

Keywords: Cryptanalysis, Fruit, Lightweight, Stream Cipher.

1 Introduction

In modern cryptography, stream ciphers play a vital role because of the need for lightweight cryptosystem. Over the last few years, lightweight stream ciphers have drawn serious attention due to its huge application in devices with limited processing power. Low area size of the cipher helps to install other protection mechanisms for the cipher. Also, it reduces the power consumption of the machine. Recent applications like WSN, RFID etc require use of lightweight ciphers.

Stream cipher is basically a class of symmetric key ciphers which generates pseudo-random keystream. It starts with a secret key and an IV. In the first phase of a stream cipher, the secret key and the IV is loaded and the Key Scheduling Algorithm is performed, without generating any output keystream. In the second phase, which is called PRGA, the keystream is generated. This keystream is directly XOR-ed with plaintext bit by bit. There are many stream ciphers which are currently being used in the market. RC4, Fish [6] are the examples of ciphers which have been immensely used in the last decade. However, most of these ciphers showed severe insecurity in recent times. At the same time, lightweight ciphers are in high demand from industries. As a result, many new ciphers has been proposed in last decade.

Grain [14], Led [13], Twine [21], Lblock [22], Present [7], Ktantan [9], Klein [12], Trivium [8], Clefia [20] etc. are some of the promising new ciphers.

A project named NESSIE was arranged in 1999 in search of some new ciphers for future adoption. Unfortunately it did not achieved much success as most of proposed ciphers fail to stand against cryptanalysis. This led to the launch of eSTREAM by EU ECRYPT in 2004. This multi-year project had two portfolios, namely hardware and software. Total 34 ciphers were proposed in the first phase of eSTREAM, among which only a few made it to the next phases of the project. After the last revision, in hardware category, only Mickey [2], Grainv v1 and Trivium [8] are still the part of this portfolio. However, all these ciphers use a large internal state to generate the keystream bits. The design of lightweight stream cipher requires the reduction of the internal state. Unfortunately, for the usual design structure of stream cipher, to resist the Time-Memory-Data tradeoff attack [4, 5], the common principle is to keep the internal state size at least twice the key size. This made the construction of more lightweight stream ciphers challenging.

In 2015, Armknetcht et al. [1] suggested a slight change in the basic design of stream ciphers to reduce the internal state without harming its security against TMD trade-off attack. While usually the secret key is involved in the initialisation process only, they suggested to use it repeatedly while encryption. In a cipher, initially the key is stored in a Non-Volatile memory (NVM), which means, the key values in this locations do not change while the cipher is running. After the process starts, the key is loaded into Volatile Memory and used. After the use of the key, the values of Volatile Memory changes, but the key is still stored in Non-Volatile memory. In classical design of stream cipher, the NVM is of no use after this initialisation. This is where [1] made the change in design. According to the new idea of Armknetcht et al. [1], unlike the classical stream cipher, the involvement of keys is not finished after the initialisation process. Rather, in each clock, the key is loaded from the NVM to VM and used in keystream generation. Since the key is already stored in NVM, the design doesnot require any extra memory to store the key. Based on this, they proposed a new cipher, named Sprout, which used 80-bit key, while its internal state size was also 80. This new idea attracted cryptographers. Surprisingly, Sprout was not secure. Lallemand et al. [16] attacked the cipher with complexity around 2^{70}, which was 2^{10} times faster than the exhaustive key search. This attack was based on a divide and conquer method. In [17], Maitra et al. provided a fault attack on Sprout. Then, Esgin et al. [10] presented a trade-off attack with time complexity 2^{33} and using memory of 770 terabytes. In Indocrypt 2015, Banik [3] attacked Sprout with low data complexity. In Asiacrypt 2015, Zhang et al [23] attacked Sprout with complexity 2^{20} times less than [16].

Despite the attacks against Sprout, the new idea used in it showed the possibility to design stream ciphers with significantly low internal state size. Based on the same idea, several new ciphers has been proposed recently. In 2016, Hamann et al. [15] presented Lizard, which used 120 bit key and 121 bit inner state. Mikhalev et al. proposed a cipher named Plantlet [18]. It uses 80 bit key, and the LFSR and NFSR sizes are 61 and 40 respectively.

Fruit is also a cipher inspired by the same idea of repeated use of the key. Ghafari, Hu and Chen designed this new ultra-lightweight cipher [11]. Its internal state size is 80, which is same as the key size. To resist the attack ideas proposed against Sprout, Ghafari et al. used some new techniques. Most of attacks used against Sprout concerned about the bias of the round key function. To protect Fruit from these attacks, they used a different and more

complicated round key function. A larger LFSR has been used. They also prevented the LFSR bits to become all zero after initialisation.

According to the authors [11], Fruit is much more secure than ciphers like Grain and Sprout against the cube attack, TMD trade-off attack etc. It also has no weak key-IV. Authors also compared its area size and hardware implementation results with Sprout and Grain. These comparisons show that Fruit is much more lightweight than those ciphers. As given in [11], the area size of Grain is around 20% more than Fruit, and gate equivalents used by Fruit is less than 80% of that of Grain. In this paper we cryptanalyse Fruit. We present an attack which is inspired by divide-and-conquer idea of [16] against Sprout. Our attack recovers the whole key in complexity $2^{74.95}$.

Paper Organization:

- In section 2, we discuss the structure of the cipher Fruit, which is pretty similar to Grain.
- Section 3, we discuss our cryptanalysis in detail. We attack the full round cipher in two phases. Section 3.1 discusses the first phase of our attack, which consists of two types of sieving: 1-bit sieving and probabilistic sieving.
- In section 3.2, we discuss the second phase of our attack, which concerns about pruning the possible states by the equations formed by keystream output bits. Using this method along with the sieving in Section 3.1, our attack complexity becomes significantly lower than the exhaustive search.

2 Description of Fruit

Here we briefly describe the design of Fruit. The designers of this cipher aim at keeping the size of the state small, but at the same time oppose the time-data-memory tradeoff attack. Here, the internal state is of 80 bits, which is same as the size of the secret key. It is composed of an LFSR of 43 bits and NFSR of 37 bits. With the 80-bit secret key, an IV of 70 bit is also given as input. Under a single IV, maximum 2^{43} keystreams can be produced. For security, the authors also suggest to use each key less than 2^{15} times with different IV and not to reuse the same IV with different keys. At first, we provide some common notations:

1. t: the clock number.
2. L_t: the LFSR state $(l_t, l_{t+1}, l_{t+2}, \cdots, l_{t+42})$ at clock t.
3. N_t: the NFSR state $(n_t, n_{t+1}, n_{t+2}, \cdots, n_{t+36})$ at clock t.
4. Cr : 7-bit counter $(c_t^0, c_t^1, c_t^2, \cdots, c_t^6)$.
5. Cc : 8-bit counter $(c_t^7, c_t^8, c_t^9, \cdots, c_t^{14})$.
6. k: the secret key $(k_0, k_1, \cdots, k_{79})$.
7. k'_t: the round key bit generated at clock t.
8. $IV = (v_0, v_1, \cdots, v_{69})$
9. z_t: keystream bit generated at clock t.

Counters: Unlike most of the other similar cipher, Fruit breaks its 15 bit counter into two parts. The first part (Cr) is of 7-bit. It is allocated to round key generation. The next 8 bits (Cc) are used in keystream generation. Both these counter starts from 0 and increases by one at eack clock. These two counters are independent.

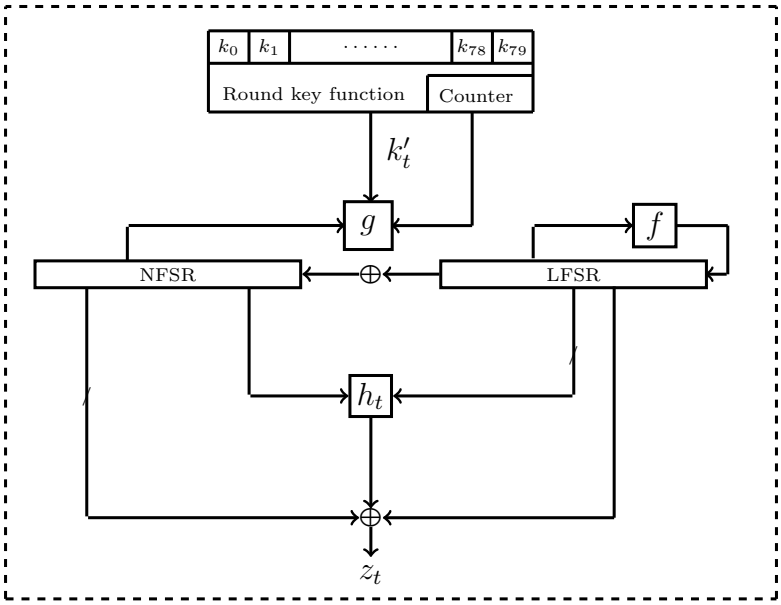

Fig. 1. Structure of Stream Cipher Fruit

Round key function: The bits of round key is generated using 6 bits of the key, by the following function:

$$k'_t = k_s k_{(y+64)} \oplus k_{(u+72)} k_p \oplus k_{(q+32)} \oplus k_{(r+64)}.$$

Here, the values of s, y, u, p, q, r are given as $s = (c_0^t c_1^t c_2^t c_3^t c_4^t c_5^t), y = (c_3^t, c_4^t, c_5^t), u = (c_4^t, c_5^t, c_6^t)$, $p = (c_0^t c_1^t c_2^t c_3^t c_4^t), q = (c_1^t c_2^t c_3^t c_4^t c_5^t), r = (c_3^t c_4^t c_5^t c_6^t)$.

LFSR: The LFSR is of 43 bits. The feedback rule of LFSR is

$$l_{(t+43)} = f(L_t) = l_t \oplus l_{t+8} \oplus l_{(t+18)} \oplus l_{(t+23)} \oplus$$

$$l_{(t+28)} \oplus l_{(t+37)}.$$

NFSR: In Fruit, the length of NFSR is 37 bits. The feedback function uses a counter bit of Cc, k'_t, l_t and a non-linear function g over N_t. Here

$$n_{t+37} = g(N_t) \oplus k'_t \oplus l_t \oplus c_t^{10},$$

where g is given by

$g(N_t) = n_t \oplus n_{t+10} \oplus n_{t+20} \oplus n_{t+12} n_{t+3} \oplus n_{t+14} n_{t+25} \oplus n_{t+5} n_{t+23} n_{t+31} \oplus n_{t+8} n_{t+18} \oplus n_{t+28} n_{t+30} n_{t+32} n_{t+34}$.

Output function: The computation of the output bit z_t is done applying a function over few selected bits of NFSR and LFSR. 1 bit of LFSR and 7 bits of NFSR are XORed with the

value of a non-linear function h over LFSR and NFSR. Output bit is $z_t = h_t \oplus n_t \oplus n_{t+7} \oplus n_{t+13} \oplus n_{t+19} \oplus n_{t+24} \oplus n_{t+29} \oplus n_{t+36} \oplus l_{t+38}$, where $h_t = n_{t+1}l_{t+15} \oplus l_{t+1}l_{t+22} \oplus n_{t+35}l_{t+27} \oplus n_{t+33}l_{t+11} \oplus l_{t+6}l_{t+33}l_{t+42}$.

Initialisation: 1 bit one and 9 bit zeros are concatenated at the beginning of the IV. Also 50 bit zeros are concatenated at the end. So, the IV is of 130 bits (we call it IV'), which looks like:

$$IV' = 1000000000v_0v_1 \cdots v_{68}v_{69}000 \cdots 00.$$

At the initial stage, the 80 key bits are loaded in the LFSR and NFSR. The first 37 key bits are loaded in NFSR and the remaining 43 keybits are loaded in LFSR. All c_i's are taken as 0 initially. In the first stage, the cipher is clocked 130 times, and the keystream bits z_t's are not given as output. Rather it is XOR-ed with the IVs and then fed to both NFSR and LFSR. So,

$$l_{t+43} = z_t \oplus v'_t \oplus f(L_t)$$

$$n_{t+37} = z_t \oplus v'_t \oplus g(N_t) \oplus k'_t \oplus l_t \oplus c_t^{10}.$$

In second stage, set all bits of Cr equal to LSB of the NFSR except the last bit that is equal to LSB of the LFSR. Also set $l_{130} = 1$. In this stage, $z_t \oplus v_t$ is not fed to the NFSR and LFSR. The cipher still do not give z_t as output.

Keystream generation: At the end of 210 rounds of first and second stage (130+80=210), the cipher starts generating output. This output z_t is XORed with the plaintext to get the ciphertext.

Inverse Operation: Suppose at any clock t, the state (L_t, N_t) is known. To find (L_{t-1}, N_{t-1}), we only need the values of l_{t-1} and n_{t-1}, which can be given as follows:

$l_{t-1} = l_{(t+42)} \oplus l_{(t+7)} \oplus l_{(t+17)} \oplus l_{(t+22)} \oplus l_{(t+27)} \oplus l_{(t+36)}$

$n_{t-1} = n_{t+36} \oplus k'_{t-1} \oplus l_{t-1} \oplus c_{t-1}^{10} \oplus n_{t+9} \oplus n_{t+19} \oplus n_{t+11}n_{t+2} \oplus n_{t+13}n_{t+24} \oplus n_{t+4}n_{t+22}n_{t+30} \oplus n_{t+7}n_{t+17} \oplus n_{t+27}\ n_{t+29}n_{t+31}n_{t+33}$

3 Key recovery attack on Fruit

In this section, we describe an attack for full round Fruit. First phase of our attack is based on divide and conquer approach. Using this idea, we reduce our search space. In the next phase, we prune further by using a clever guess and determine approach. Let us start with simple observations.

Linear register: Note that the linear register state is totally independent from the rest during the keystream generation phase. Thus, once its 43-bit value at time t are guessed, we can compute all of its future states during the keystream generation.

Counter: After 130 rounds of the initialisation process, first part of the counter Cr is fed krom the LFSR and NFSR. Thus after 130 rounds, attacker does not know Cr. But the second part of the counter Cc is deterministic and it is independent of the key. So c_t^{10}, used in NFSR feedback, is known to the attacker.

3.1 First phase of the attack

At any time t in the keystream generation process, we guess the state. Since total size of LFSR and NFSR is $43+37=80$, number of possible state size is 2^{80}. Now we reduce this size using two ideas. Our first idea is deterministic in nature whereas is second one is probabilistic. Let us start with first idea.

Sieving of 1 bit: Let us consider the output function $z_t = n_{t+1}l_{t+15} \oplus l_{t+1}l_{t+22} \oplus n_{t+35}$ $l_{t+27} \oplus n_{t+33}l_{t+11} \oplus l_{t+6}l_{t+33}l_{t+42} \oplus n_t \oplus n_{t+7} \oplus n_{t+13} \oplus n_{t+19} \oplus n_{t+24} \oplus n_{t+29} \oplus n_{t+36} \oplus l_{t+38}$. We can see that no key bit has been used to compute z_t. So, at any clock t, if we know the internal state, we can compute the output z_t, without knowing any key bit (see figure 2). As z_t is already known, this gives us sieving of one bit, i.e, only by knowing 79 bits of the state, we can compute the remaining bit from z_t. So, the number of possible state candidates is reduced by half i.e, 2^{79}.

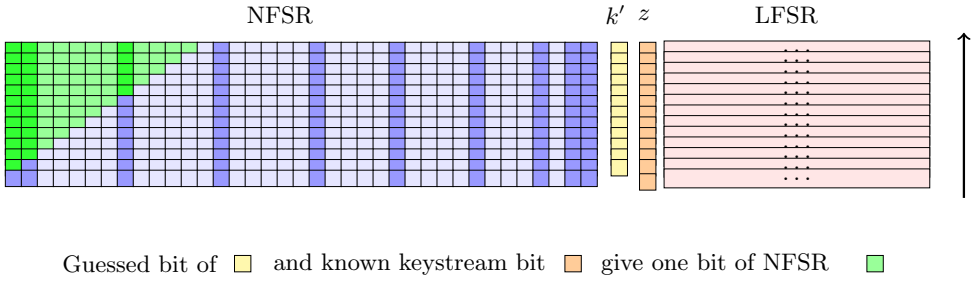

Fig. 2. Our attack idea on Fruit.

Probabilistic Sieving-bias of k'_t : In Fruit, round key k'_t is the combination of 6 key bits. However, we observe a positive bias of k'_t towards 0. The value of k'_t depends on the counter value Cr by

$$k'_t = k_s k_{y+64} \oplus k_{(u+72)}k_p \oplus k_{(q+32)} \oplus k_{(r+64)}.$$

Now, for some values of Cr, k'_t gives 0 with high probability. For example, if Cr is 1111111, $k'_t = k_{63}k_{71} \oplus k_{79}k_{31} \oplus k_{63} \oplus k_{79} = k_{63}(k_{71} \oplus 1) \oplus k_{79}(k_{31}+1)$. Among 16 possible values of $(k_{63}, k_{71}, k_{79}, k_{31})$, only six give $k'_t = 1$. So, $\Pr(k'_t = 0) = \frac{5}{8}$. We observe that among 128 possible values of Cr, there are 32 such values for which this probability is $\frac{5}{8}$, whereas only for four values this probability is $\frac{3}{8}$. Remaining all counter values give probability $\frac{1}{2}$. Overall, k'_t takes value 0 with probability $\frac{135}{256} = 52.7\%$. In the following table, we give the counter values in decimal form for which the probabilities are less than or greater than $\frac{1}{2}$.

Reducing complexity using this bias: Note that if we know k'_t, we can find n_{t+37} from the knowledge of z_t. See figure 2. This reduces the guessed size of the state using sieving. Since k'_t is biased towards 0, we use this fact to guess the value of k'_t. Suppose, we want to

Counter with $\Pr(k'_t = 0) = \frac{3}{8}$	Counter with $\Pr(k'_t = 0) = \frac{5}{8}$
64	72-79
80	88-95
96	104-111
112	120-127

Table 1. Distribution of k'_t for different counter values.

guess the values of $k'_{t-1}, k'_{t-2}, \cdots, k'_{t-r}$. We start with the strings which contains more 0's. It gives high probability to make our guess correct. For example, when $r = 4$, the string 0000 has probability $(0.527)^4\%$ to be correct, where as the probability of 1111 is $(0.473)^4\%$ if the events $k'_i = 0$ and $k'_j = 0$ are independent for $i \neq j$.

So, while guessing an r-bit string for $k'_{t-1}, k'_{t-2}, \cdots k'_{t-r}$, we take our first guess as $\overbrace{00\cdots0}^{r}$. Our next guess would be a string containing one bit 1 and $r-1$ bits 0's. After guessing all such strings, we go for the strings containing two bits 1's and $r-2$ bits 0's and so on. Thus we go on increasing the number of 1's and decreasing the number of 0's in the string. Suppose, X_r is a random variable which denotes the number of guesses required to find the correct $k'_{t-1}, \cdots, k'_{t-r}$. Now expected of value of X_r is

$$E(X_r) = \sum_{i=0}^{2^r-1} i \left(\frac{135}{256}\right)^{\text{count}_{(i,0)}} \left(\frac{121}{256}\right)^{r-\text{count}_{(i,0)}},$$

where $\text{count}_{(i,0)}$ denotes the number of 0's when i is represented as a binary r bit number.

Since $E(X_r) < 2^{r-1}$, the complexity of the search is reduced by $\frac{2^r}{E(X_r)}$. In Table 2, we give the expected reduction for different r's. As we see from the table, increasing the value of r increases the reduction factor. When $r = 12$, the reduction factor almost reaches a constant value.

r	6	8	10	12	14
$E(X_r)$	30.777	121.527	484.527	1936.527	7744.526
Reduction factor	2.079	2.107	2.113	2.115	2.116

Table 2. Reduction factor for different r consecutive guesses.

In Table 2, we assume that the events $\{k'_{t-i} = 0\}$ and $\{k'_{t-j} = 0\}$ are independent for $1 \leq i, j \leq r$ with $i \neq j$. But this not the case. There are 32 counter values for which $P(k'_t = k'_{t-1}) = \frac{3}{4}$. Similarly there are 16 counter values for which $P(k'_t = k'_{t-2}) = \frac{9}{16}$. So k_{t-i} and k_{t-j} are highly correlated for $i \neq j$. Thus we consider an ordering for guessing

$k'_{t-1}, k'_{t-2}, \cdots, k'_{t-r}$. Experiment with 2^{20} many random keys shows that

$$P\Big((k'_{t-1}, \cdots, k'_{t-12}) = (\overbrace{0, \ldots, 0}^{12})\Big) = 3.70 \times \left(\frac{135}{256}\right)^{12}.$$

That is the probability $(k'_{t-1}, \cdots, k'_{t-12}) = (\overbrace{0, \ldots, 0}^{12})$ is 3.70 times more than our initial prediction $\left(\frac{135}{256}\right)^{12}$. In Table 3, we present actual reduction factor. Experiment is done over random 2^{27} random keys.

r	6	8	10	12
$E(X_r)$	26.622	100.377	382.818	1465.256
Reduction factor	2.404	2.550	2.675	2.795
r	14	16	18	20
$E(X_r)$	5623.352	21625.896	79072.808	277169.368
Reduction factor	2.913	3.030	3.315	3.783

Table 3. Actual reduction factor for different r consecutive guesses.

Building List for LFSR and NFSR: To reduce the complexity, we construct separate lists for all possible values of LFSR and NFSR. At any t_0-th clock, instead of guessing the whole 80 bits of the internal state, if we guess the LFSR bits and NFSR bits separately, the complexity decreases significantly. For this purpose, we build independent lists for LFSR and NFSR, which we denote as $\mathcal{L}_L$ and $\mathcal{L}_N$ respectively. Since the LFSR is of length 43, $\mathcal{L}_L$ contains all 2^{43} possible state values of LFSR. For NFSR, the list $\mathcal{L}_N$ contains few more values except the 2^{37} possible state values. Let us have a look at the inverse NFSR function:

$n_{t-1} = n_{t+36} \oplus k'_{t-1} \oplus l_{t-1} \oplus c^{10}_{t-1} \oplus n_{t+9} \oplus n_{t+19} \oplus n_{t+11}n_{t+2} \oplus n_{t+13}n_{t+24} \oplus n_{t+4}n_{t+22}n_{t+30} \oplus n_{t+7}n_{t+17} \oplus n_{t+27}n_{t+29}n_{t+31}n_{t+33}$.

Here, the term c^{10}_{t-1} is known. But k'_{t-1} and l_{t-1} are unknown. So, we include a new column for k'_t in $\mathcal{L}_N$. Similarly, since we have to consider both possible values of l_t (0 and 1), we create another column for l_t . Let us consider $r+1$ consecutive backward rounds, beginning from the clock t_0, i.e, rounds $t_0, t_0-1, t_0-2, \cdots, t_0-r$. Then, in each of these rounds, k'_t is unknown. One can not compute the keystream bit z_{t-1} from the knowledge of state (L_t, N_t) without knowing k'_{t-1}. For this reason, to sieve further we first guess the values $k'_{t_0-1}, k'_{t_0-2}, \cdots, k'_{t_0-r}$ and use the knowledge of keystream bits $z_{t_0-1}, \cdots, z_{t_0-r}$. So, we have to consider both values (0 and 1) each of $k'_{t_0-1}, k'_{t_0-2}, \cdots, k'_{t_0-r}$, which gives 2^r possible cases. But one can see from the Table 3 that to find the correct value of the tuple $(k'_{t_0-1}, k'_{t_0-2}, \cdots, k'_{t_0-r})$, average required guess is much smaller than 2^r.

For each such case, we consider 2^{43} possible internal LFSR states and 2^{37} possible NFSR states. These lists $\mathcal{L}_L$ and $\mathcal{L}_N$ are also sorted according to the values of some expressions. Detail explanation of this is given later.

Detail Explanation of Sieving of 1 bit using Lists $\mathcal{L}_L$ and $\mathcal{L}_N$: Here we give a detail explanation on how to construct the the lists $\mathcal{L}_L$ and $\mathcal{L}_N$ and reduce the complexity to half. Since we do not consider the probabilistic sieving here, we do not use the columns for k'_t. So, we assume $\mathcal{L}_N$ to have 2^{37} possible state values only. If we look at the function $z_t = n_{t+1}l_{t+15} \oplus l_{t+1}l_{t+22} \oplus n_{t+35}l_{t+27} \oplus n_{t+33}l_{t+11} \oplus l_{t+6}l_{t+33}l_{t+42} \oplus n_t \oplus n_{t+7} \oplus n_{t+13} \oplus n_{t+19} \oplus n_{t+24} \oplus n_{t+29} \oplus n_{t+36} \oplus l_{t+38}$,

we can divide the terms into three types

1. Terms having only LFSR bits.
2. Terms having only NFSR bits.
3. Terms having both NFSR and LFSR bits.

We denote the sum of the terms involving LFSR bits and z_t by τ_l and the sum of the terms involving NFSR bits by τ_n. So, $\tau_l = l_{t+1}l_{t+22} \oplus l_{t+6}l_{t+33}l_{t+42} \oplus l_{t+38} \oplus z_t$ and $\tau_n = n_t \oplus n_{t+7} \oplus n_{t+13} \oplus n_{t+19} \oplus n_{t+24} \oplus n_{t+29} \oplus n_{t+36}$. So, we have $\tau_l \oplus \tau_n \oplus n_{t+1}l_{t+15} \oplus n_{t+35}l_{t+27} \oplus n_{t+33}l_{t+11} = 0$. Among LFSR bits, l_{t+15}, l_{t+27} and l_{t+11} are involved in product with LFSR bits. So, once we have the value of z_t, we sort our $\mathcal{L}_L$ list by the values of $\tau_l, l_{t+15}, l_{t+11}, l_{t+27}$. For this, consider the tuple $(\tau_l, l_{t+15}, l_{t+11}, l_{t+27})$. This tuple can have 16 possible values. Based on this values, we divide the list $\mathcal{L}_L$ into 16 parts. Similarly, we sort $\mathcal{L}_N$ according to $\tau_n, n_{t+33}, n_{t+35}, n_{t+1}$. So, using the equation $\tau_l \oplus \tau_n \oplus n_{t+1}l_{t+15} \oplus n_{t+35}l_{t+27} \oplus n_{t+33}l_{t+11} = 0$, we can choose our LFSR and NFSR states by gradual matching [19]. This will help us to reduce the complexity by half.

We give a simple example. Suppose we are checking the states for which l_{t+15}, l_{t+11} and l_{t+27} is zero. Then, to satisfy the equation $\tau_l \oplus \tau_n \oplus n_t l_{t+15} \oplus n_{t+35}l_{t+27} \oplus n_{t+33}l_{t+11} = 0$, we combine the part of $\mathcal{L}_L$ for which $\tau_l = 0$ only with the parts of $\mathcal{L}_N$ for which $\tau_n = 0$, and not with the parts for which $\tau_n = 1$. Similarly, with the LFSR states where $\tau_l = 1$, we combine the NFSR states for which $\tau_n = 1$. In particular, the part $\mathcal{L}_{L(0,0,0,0)}$ is combined with $\mathcal{L}_{N(1,1,1,0)}$ but not with $\mathcal{L}_{N(1,1,1,1)}$, whereas for $\mathcal{L}_{L(0,0,0,0)}$ we do the opposite.

Probabilistic Sieving Using lists $\mathcal{L}_L$ and $\mathcal{L}_N$: We can perform probabilistic sieving using the two lists $\mathcal{L}_L$ and $\mathcal{L}_N$. In this case, the attacker is provided a number of consecutive keystream bits and she has to guess the value of k'_t for those clocks. Now, instead of guessing k'_t values arbitrarily, following a pattern will help to find the correct guess with less number of attempts, as mentioned before.

Suppose, the attacker tries to find the state (L_{t_0}, N_{t_0}) at time $t = t_0$. She will use the keystream bits $z_{t_0}, z_{t_0-1}, \cdots z_{t_0-r}$ for this. For a state (L_{t_0}, N_{t_0}), the z_{t_0} can be computed directly, without any value of k'_t. But subkey k'_{t_0-1} is involved in z_{t_0-1} because it depends on n_{t_0-1}. Again n_{t_0-1} depends on k'_{t_0-1} and l_{t_0-1}. So, the attacker has to guess $k'_{t_0-1}, k'_{t_0-2}, \cdots k'_{t_0-r}$. Now, instead of considering all possible values for the tuple $(k'_{t_0-1}, k'_{t_0-2}, \cdots, k'_{t_0-r})$, we arrange the values according to their probabilities. Let the expected number of attempts to find $(k'_{t_0-1}, k'_{t_0-2}, \cdots, k'_{t_0-r})$ be $E(X_r)$. We consider the first $E(X_r)$ number of choices for $(k'_{t_0-1}, k'_{t_0-2}, \cdots, k'_{t_0-r})$. For each such values we have to consider the 2^{37} NFSR states. Again, since at each round $(t_0 - j)$ ($j = 1$ to r), we need to consider both possible values for l_{t_0-j}. So, the final size of the list $\mathcal{L}_N$ is $2^{37} \cdot E(X_r) \cdot 2^r = 2^{37+r} \cdot E(X_r)$. After that, the list is sorted in the following way.

1. Based on the output keystream z_{t_0} at the first round, at first we sort the list just the same way as we mentioned in 1-bit sieving. $\mathcal{L}_L$ is sorted based on the values of $\tau_{l_0} = (l_{t_0+1}l_{t_0+22} \oplus l_{t_0+6}l_{t_0+33}l_{t_0+42} \oplus l_{t_0+38} \oplus z_{t_0}), l_{t_0+15}, l_{t_0+11}, l_{t_0+27}$ and $\mathcal{L}_N$ is sorted according to $\tau_{n_0} = (n_{t_0} \oplus n_{t_0+7} \oplus n_{t_0+13} \oplus n_{t_0+19} \oplus n_{t_0+24} \oplus n_{t_0+29} \oplus n_{t_0+36}), n_{t_0+33}, n_{t_0+35}, n_{t_0+1}$. So, $\mathcal{L}_L$ and $\mathcal{L}_N$, both are divided into 2^4 sublists. Now among this, only $2^{(4+4-1)} = 2^7$ combinations are eligible for correct state, by 1-bit sieving. For each possible combination of sublists from $\mathcal{L}_L$ and $\mathcal{L}_N$, we consider the next j.
2. For all $t = t_0 - 1$ to $t_0 - r$, we further subdivide (sort) the sublists formed in previous step. Based on the value of z_t, $\mathcal{L}_L$ is sorted according to $\tau_l, l_{t+15}, l_{t+11}, l_{t+27}$ and also l_t. Also $\mathcal{L}_N$ is sorted according to $\tau_n, n_{t+33}, n_{t+35}, n_{t+1}$. While merging, corresponding to the LFSR states where $l_t = 0$, we consider the NFSR states in $\mathcal{L}_N$ with $l_t = 0$. Same we do with $l_t = 1$. Again, for each possible combination of sublists from $\mathcal{L}_L$ and $\mathcal{L}_N$, we consider the previous clock $t - 1$.

Thus, due to direct 1 bit sieving and probabilistic sieving, total number of possible state will be now $\frac{2^{80}}{2^{r+1}} = 2^{79-r}$. If we guess $(k'_{t_0-1}, k'_{t_0-2}, \cdots k'_{t_0-r})$ for $E(X_r)$ times, total number of possible state is $2^{79-r} \times E(X_r)$. In Table 4, values $2^{79-r} \times E(X_r)$ are presented for different r.

r	10	12	14	16	18	20
$2^{79-r} \times E(X_r)$	$2^{77.58}$	$2^{77.52}$	$2^{77.46}$	$2^{77.40}$	$2^{77.27}$	$2^{77.08}$

Table 4. Size of the final possible state for different r.

3.2 Second phase of the attack: Guessing a middle state

Using our first approach, we have a total of $2^{77.08}$ possible states. Now problem is how we can reduce this size further? At any time $t_0 > 0$, we guess an 80-bit vector for the internal state (L_{t_0}, N_{t_0}). Now, since $z_{t+1} = h_{t+1} \oplus n_{t+1} \oplus n_{t+8} \oplus n_{t+14} \oplus n_{t+20} \oplus n_{t+25} \oplus n_{t+30} \oplus n_{t+37} \oplus l_{t+39}$, we have $n_{t+37} = h_{t+1} \oplus n_{t+1} \oplus n_{t+8} \oplus n_{t+14} \oplus n_{t+20} \oplus n_{t+25} \oplus n_{t+30} \oplus z_{t+1} \oplus l_{t+39}$, which we compute using the output bit z_{t+1} and the guessed NFSR and LFSR bits. Now, from the equations $n_{t+37} = g(N_t) \oplus k'_t \oplus l_t \oplus c_t^{10}$ and $k'_t = k_s k_{(y+64)} \oplus k_{(u+72)} k_p \oplus k_{(q+32)} \oplus k_{(r+64)}$, we form the equation $k_s k_{(y+64)} \oplus k_{(u+72)} k_p \oplus k_{(q+32)} \oplus k_{(r+64)} \oplus g(N_t) \oplus l_t \oplus c_t^{10} \oplus n_{t+37} = 0$. Expressing the sum $g(N_t) \oplus l_t \oplus c_t^{10} \oplus n_{t+37}$ as α_t, the equation becomes $k_s k_{(y+64)} \oplus k_{(u+72)} k_p \oplus k_{(q+32)} \oplus k_{(r+64)} \oplus \alpha_t = 0$. For, any $t \geq t_0$, this equation should be satisfied jf the guess of internal state (L_{t_0}, N_{t_0}) is correct. Now, we observe experimentally that for random counter, we can discard 52% wrong state if we take 24 keystream bits. Experiment is done over 100000 random key-IV.

In Appendix A, we explain it by an example for the counter $Cr = 0$. We show that for the counter $Cr = 0$, 24 key stream bits are sufficient to discard around 60% wrong guessed states.

Average number of output keystream bits required to eliminate a wrong state: We give an approximate measure of the average number of z_i's required to discard an incorrect state. A set of 24 keystream bits can discard at least 50% wrong guesses. Now, suppose, total number of possible guesses are S. For 24 clocks (24 z_i's), on average, around $\frac{S}{2}$ guesses will be eliminated. From the remaining $\frac{S}{2}$ guesses, again half of them will be eliminated when we use the equations derived from next 24 z_i's. So, we are left with $\frac{S}{4}$ guesses. In this way, in general for any i, around $\frac{S}{2^i}$ guesses are there which do not eliminate for first $24(i-1)$ equations, but eliminates when we use $24i$ equations. For these many guesses, the number of z_i's required for elimination is $24i$. To calculate the average number of clocks required for elimination, we compute $\sum_i \frac{S}{2^i} \times 24$ and divide by S. So, required average is

$$\frac{\sum_i \frac{S}{2^i} \times 24}{S} \approx \frac{2S \times 24}{S} \approx 48$$

Based on this idea, we construct some tables in preprocessing phase.

Construction of Table:

In the preprocessing phase, we construct ℓ tables, say Table $T_1, T_2, \cdots, T_\ell$. In each table, the first column contains all possible 24 bit binary strings, in increasing order. There are total 2^{24} such strings. In the first table (T_1), each of the strings denotes a possible sequence of $\alpha_i\alpha_{i+1}\cdots\alpha_{i+23}$. Corresponding to each such string $\alpha_i\alpha_{i+1}\cdots\alpha_{i+23}$, in the second column we record the possible values of Cr at i-th clock. There are at most 128 values of Cr. So, the data complexity of this table is 2^{31}. Similarly, in table T_2, the first column contains all possible 24 bit binary string, which corresponds to $\alpha_{i+24}\alpha_{i+25}\cdots\alpha_{i+47}$. In the second column, for each string $\alpha_{i+24}\alpha_{i+25}\cdots\alpha_{i+47}$, we record the possible values of Cr at i-th clock. We do the same for all ℓ tables.

Processing Phase:

1. After guessing a middle state at t_0-th clock, we compute the α_i's from the output keystream z_i's. We compare $\alpha_{t_0}\alpha_{t_0+1}\alpha_{t_0+2}\cdots\alpha_{t_0+23}$ with the strings in the first column by binary search to find the exact match and check the corresponding possible Cr values in the second column. Let us denote the set of these Cr values as $T_1(Cr)$.
2. If $T_1(Cr) = \phi$, we discard the state. Otherwise we go to the next table T_2. We match $\alpha_{i+24}\alpha_{i+25}\cdots\alpha_{i+47}$ with the string in the first column and check the corresponding Cr values. We denote this set of Cr values as $T_2(Cr)$. If $T_1(Cr) \cap T_2(Cr) = \phi$, we discard the state. Otherwise we go to table T_3 and find $T_1(Cr) \cap T_2(Cr) \cap T_3(Cr)$. We do the same for each table T_i. If at any table the intersection of $T_i(Cr)$s become ϕ, we discard the state.
3. If $T(Cr) = \cap_{i=1}^{\ell} T_i(Cr) \neq \phi$, we list the state along with its $T(Cr) = \cap_{i=1}^{\ell} T_i(Cr)$. So, at the end of this, we have a list of possible candidates for the state (L_t, N_t) and for each such candidate we have a set possible counter (Cr) values. Since on average 48 key stream bits are sufficient to discard a wrong state, average number of required tables is $\ell = 2$.

Time complexity of our attack: After 1 bit sieving and probabilistic sieving we have total $2^{77.08}$ possible states. Then to discard a wrong state, we have to run Fruit 48 rounds. Thus our total time complexity is $2^{77.08} \times 48$, which is equivalent to $2^{77.08} \times 48 \times \frac{1}{210} = 2^{74.95}$ many Fruit

encryption. This is 16.56 times faster than the average exhaustive search attack complexity 2^{79} Fruit encryption. Here we do not consider the following 80 rounds in the exhaustive search for recovering one unique key.

4 Conclusion

In this paper, we presented an attack 16.95 times faster than average exhaustive search. Our idea mostly uses the biases of the round keys, as well as some other pruning techniques. Though the structure of Fruit shows the possibility of designing ultra-lightweight ciphers with promising security, this attack shows that the function of round key generation in Fruit is not perfect and should be analyzed more carefully to design a cipher with such small internal state. Our attack is the first proposed attack against this cipher till date. The most important part in the design of lightweight ciphers with short internal state is the round key generation and our attack shows that one should be very careful in such cipher design.

References

1. F. Armknecht and V. Mikhalev. On Lightweight Stream Ciphers with Shorter Internal States. FSE 2015, pages: 451-470.
2. S. Babbage and M. Dodd. The MICKEY Stream Ciphers. New Stream Cipher Designs - The eSTREAM Finalists, pages: 191-209, 2008.
3. S. Banik. Some Results on Sprout. INDOCRYPT 2015, pages: 124-139, 2015.
4. E. Barkan, E. Biham and A. Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. CRYPTO 2006, pages: 1–21, 2006.
5. A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. ASIACRYPT 2000, pages: 1–13, 2000.
6. U. Blöcher and M. Dichtl. Fish: A Fast Software Stream Cipher. Fast Software Encryption 1993. http://dblp.uni-trier.de/rec/bib/conf/fse/BlocherD93
7. A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. CHES 2007, pages: 450–466, 2007.
8. C. D. Cannière and B. Preneel. Trivium. New Stream Cipher Designs - The eSTREAM Finalists. pages: 244-266, 2008.
9. C.D. Cannière, O. Dunkelman, M. Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. CHES 2009, pages: 272–288, 2009.
10. M. F. Esgin and O. Kara. Practical Cryptanalysis of Full Sprout with TMD Tradeoff Attacks. SAC 2015, pages: 67-85, 2015.
11. V. A. Ghafari, H. Hu and Y. Chen. Fruit: ultra-lightweight stream cipher with shorter internal state. IACR 2016. `http`//eprint.iacr.org/2016/355
12. Z. Gong, S. Nikova, Y.W. Law. KLEIN: A New Family of Lightweight Block Ciphers. RFIDSec 2011, pages:1–18, 2011.
13. J. Guo, T. Peyrin, A. Poschmann and M. Robshaw. The LED Block Cipher. CHES 2011, pages: 326–341, 2011.
14. M. Hell, T. Johansson and W. Meier. Grain: a stream cipher for constrained environments. IJWMC 2007, pages: 86-93. `http`//dx.doi.org/10.1504/IJWMC.2007.013798
15. M. Hamann, M. Krause and Willi Meier. LIZARD - A Lightweight Stream Cipher for Power-constrained Devices. IACR 2016. `http`//eprint.iacr.org/2016/926

16. V. Lallemand and M. N. Plasencia. Cryptanalysis of Full Sprout. CRYPTO 2015, pages: 663-682, 2015.
17. S. Maitra, S. Sarkar, A. Baksi and P. Dey. Key Recovery from State Information of Sprout: Application to Cryptanalysis and Fault Attack. IACR 2015. `http`//eprint.iacr.org/2015/236
18. V. Mikhalev, F. Armknecht and C. Müller. On Ciphers that Continuously Access the Non-Volatile Key. Accepted in FSE 2017.
19. M. N. Plasencia. How to Improve Rebound Attacks. CRYPTO 2011, pages: 188–205, 2011.
20. T. Shirai, K. Shibutani, T. Akishita, S. Moriai, T. Iwata. The 128-Bit Block- cipher CLEFIA (Extended Abstract). FSE 2007, pages: pp. 181–195, 2007.
21. T. Suzaki, K. Minematsu, S. Morioka, E. Kobayashi. TWINE : A Lightweight Block Cipher for Multiple Platforms. SAC 2012, pages: 339–354, 2012.
22. W. Wu and L. Zhang. LBlock: A Lightweight Block Cipher. Applied Cryptography and Network Security, ACNS 2011, pages: 327–344, 2011.
23. B. Zhang and X. Gong. Another Tradeoff Attack on Sprout-Like Stream Ciphers. ASIACRYPT 2015, pages: 561–585, 2015.

Appendix A:

Set of 24 equations beginning with counter $Cr = 0$:

Eq. No.	Equation	Eq. No.	Equation
1	$k_0k_{64} + k_0k_{72} + k_{32} + k_{64} + \alpha_0$	2	$k_0k_{64} + k_0k_{73} + k_{32} + k_{65} + \alpha_1$
3	$k_0k_{74} + k_0k_{65} + k_{33} + k_{66} + \alpha_2$	4	$k_0k_{75} + k_0k_{65} + k_{33} + k_{66} + \alpha_3$
5	$k_1k_{76} + k_0k_{72} + k_{32} + k_{64} + \alpha_4$	6	$k_1k_{77} + k_0k_{73} + k_{32} + k_{65} + \alpha_5$
7	$k_1k_{78} + k_0k_{65} + k_{33} + k_{66} + \alpha_6$	8	$k_1k_{79} + k_0k_{65} + k_{33} + k_{66} + \alpha_7$
9	$k_2k_{72} + k_4k_{68} + k_{36} + k_{72} + \alpha_8$	10	$k_2k_{73} + k_4k_{68} + k_{36} + k_{73} + \alpha_9$
11	$k_2k_{74} + k_5k_{69} + k_{37} + k_{74} + \alpha_{10}$	12	$k_2k_{75} + k_5k_{69} + k_{37} + k_{75} + \alpha_{11}$
13	$k_3k_{76} + k_4k_{68} + k_{36} + k_{72} + \alpha_{12}$	14	$k_3k_{77} + k_4k_{68} + k_{36} + k_{73} + \alpha_{13}$
15	$k_3k_{78} + k_5k_{69} + k_{37} + k_{74} + \alpha_{14}$	16	$k_3k_{79} + k_5k_{69} + k_{37} + k_{75} + \alpha_{15}$
17	$k_4k_{72} + k_8k_{64} + k_{40} + k_{64} + \alpha_{16}$	18	$k_4k_{73} + k_8k_{64} + k_{40} + k_{65} + \alpha_{17}$
19	$k_4k_{74} + k_9k_{65} + k_{41} + k_{66} + \alpha_{18}$	20	$k_4k_{75} + k_9k_{65} + k_{41} + k_{67} + \alpha_{19}$
21	$k_5k_{76} + k_{10}k_{66} + k_{42} + k_{68} + \alpha_{20}$	22	$k_5k_{77} + k_{10}k_{66} + k_{42} + k_{69} + \alpha_{21}$
23	$k_5k_{78} + k_{11}k_{67} + k_{43} + k_{70} + \alpha_{22}$	24	$k_5k_{79} + k_{11}k_{67} + k_{43} + k_{71} + \alpha_{23}$

We divide the first 24 equations into two disjoint sets E_1 and E_2, each containing 12 equations. In Appendix, the 24 equations are given in order. Here, we mention the equations in E_1 and from these 12 equations, we derive three conditions such that the correct candidate

cannot satisfy any one of these conditions. So, if a guessed candidate satisfies any one of these three conditions, it is not the correct candidate, and we can discard it. For convenience, here in these 12 equations of E_1, we denote the α_i's as β_j, where $j = 1$ to 12.
First set of equations:

$$k_0k_{64} + k_0k_{72} + k_{32} + k_{64} + \beta_0 \tag{1}$$

$$k_0k_{64} + k_0k_{73} + k_{32} + k_{65} + \beta_1 \tag{2}$$

$$k_0k_{74} + k_0k_{65} + k_{33} + k_{66} + \beta_2 \tag{3}$$

$$k_0k_{75} + k_0k_{65} + k_{33} + k_{66} + \beta_3 \tag{4}$$

$$k_2k_{72} + k_4k_{68} + k_{36} + k_{72} + \beta_4 \tag{5}$$

$$k_2k_{73} + k_4k_{68} + k_{36} + k_{73} + \beta_5 \tag{6}$$

$$k_2k_{74} + k_5k_{69} + k_{37} + k_{74} + \beta_6 \tag{7}$$

$$k_2k_{75} + k_5k_{69} + k_{37} + k_{75} + \beta_7 \tag{8}$$

$$k_4k_{72} + k_8k_{64} + k_{40} + k_{64} + \beta_8 \tag{9}$$

$$k_4k_{73} + k_8k_{64} + k_{40} + k_{65} + \beta_9 \tag{10}$$

$$k_4k_{74} + k_9k_{65} + k_{41} + k_{66} + \beta_{10} \tag{11}$$

$$k_4k_{75} + k_9k_{65} + k_{41} + k_{67} + \beta_{11} \tag{12}$$

Condition A:

(a) $\beta_4 + \beta_5 = 1$,
(b) $\beta_2 + \beta_3 + \beta_{10} + \beta_{11} = 0$,
(c) $\beta_0 + \beta_1 + \beta_8 + \beta_9 = 1$:

Since $\beta_4+\beta_5 = 1$, adding equations 5 and 6, we have $(k_2+1) = 1 = k_{74}+k_{75}$. We add equations 3, 4, 7, 8 and put $k_{74} + k_{75} = 1$ which together with the fact that $\beta_2 + \beta_3 + \beta_{10} + \beta_{11} = 0$ gives:$(k_0 + k_4) = 0$. Finally, adding equations 1, 2, 9, 10 and putting $(k_0 + k_4) = 0$, we have $\beta_0 + \beta_1 + \beta_8 + \beta_9 = 0$, which contradicts with (c).
Condition B:

(a) $\beta_4 + \beta_5 + \beta_6 + \beta_7 = 1$,
(b) $\beta_0 + \beta_1 + \beta_8 + \beta_9 = 1$,
(c) $\beta_2 + \beta_3 + \beta_{10} + \beta_{11} = 1$

Adding equations 5, 6, 7, 8 and from the fact that $\beta_4 + \beta_5 + \beta_6 + \beta_7 = 1$ we have $(k_2 + 1)(k_{74} + k_{75} + k_{72} + k_{73}) = 1$. This implies that $(k_{74} + k_{75} + k_{72} + k_{73}) = 1$. But, adding equations 1, 2, 9 and 10 and using $\beta_0 + \beta_1 + \beta_8 + \beta_9 = 1$, we have $k_{72} + k_{73} = 1$. Also, adding equations 3, 4, 11 and 12, we have $(k_{74} + k_{75}) = 1$. (since $\beta_2 + \beta_3 + \beta_{10} + \beta_{11} = 1$). So, we have $(k_{74} + k_{75} + k_{72} + k_{73}) = 0$, which is a contradiction.
Condition C:

(a) $\beta_6 + \beta_7 = 1$,
(b) $\beta_0 + \beta_1 + \beta_8 + \beta_9 = 0$,

(c) $\beta_2 + \beta_3 + \beta_{10} + \beta_{11} = 1$

Adding equation 7, 8 and (a) , we have $k_{72} + k_{73} = 1$. Using this fact and adding equations 1, 2, 9, 10 we have $(k_0 + k_4 + \beta_0 + \beta_1 + \beta_8 + \beta_9) = 0$. From (b), we have $k_0 + k_4 = 0$. Now, using $k_0 + k_4 = 0$ and adding equation 3, 4, 11, 12, we get $\beta_2 + \beta_3 + \beta_{10} + \beta_{11} = 0$, which contradicts (c). Let, by $\mathcal{A}, \mathcal{B}, \mathcal{C}$ we denote the set of candidates satisfying conditions A, B, C respectively. Looking at the conditions, one can easily observe that the sets $\mathcal{A}, \mathcal{B}, \mathcal{C}$ are mutually disjoint. Since each condition has three equations, the number of candidates satisfying a condition is $\frac{1}{2^3}$ fraction of total candidates. So, $|\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}|$ is $\frac{3}{8}$ fraction of the total. Similarly, for the second set of equations, we can find 3 conditions, namely condition P, Q, R, each containing three equations. So, denoting the corresponding set of candidates as $\mathcal{P}, \mathcal{Q}, \mathcal{R}$, we have $|\mathcal{P} \cup \mathcal{Q} \cup \mathcal{R}| = \frac{3}{8}$. Since the conditions A, B, C are independent from P, Q, R, we have

$$|\mathcal{A} \cup \mathcal{B} \cup \mathcal{C} \cup \mathcal{P} \cup \mathcal{Q} \cup \mathcal{R}| = \frac{3}{8} + \frac{3}{8} - \left(\frac{3}{8}\right)^2 \approx 0.60$$

fraction of total.