

# A Time-Memory Tradeoff Attack Against LILI-128

**\*\* DRAFT - October 16, 2001 \*\***

Markku-Juhani Olavi Saarinen

Nokia Research Center  
P.O. Box 407, FIN-00045 Nokia Group, Finland  
markku-juhani.saarinen@nokia.com

**Abstract.** In this note we discuss a novel but simple time-memory tradeoff attack against the stream cipher LILI-128. The attack defeats the security advantage of having an irregular stepping function. The attack requires  $2^{46}$  bits of keystream, a lookup table of  $2^{45}$  89-bit words and computational effort which is roughly equivalent to  $2^{48}$  DES operations.

## 1 Introduction

The LILI-128 keystream generator [1] is a LFSR-based synchronous stream cipher with a 128 bit key. It has been accepted as one of six candidate stream ciphers for NESSIE.

In the original LILI-128 specification, the authors conjecture that the complexity of divide and conquer attacks is “at least  $2^{112}$  operations, requiring knowledge of at least 1700 known keystream bits”.

After its initial release, some cryptanalytic results on LILI-128 has been published [2, 5]. The best known attacks are:

- In [3], Jönsson and Johansson describe an attack of complexity  $2^{79}$ , requiring  $2^{30}$  keystream bits and a off-line precomputed table with  $2^{79}$  entries.
- In [4], Babbage discusses a rekeying attack and generic time-memory tradeoff attacks.

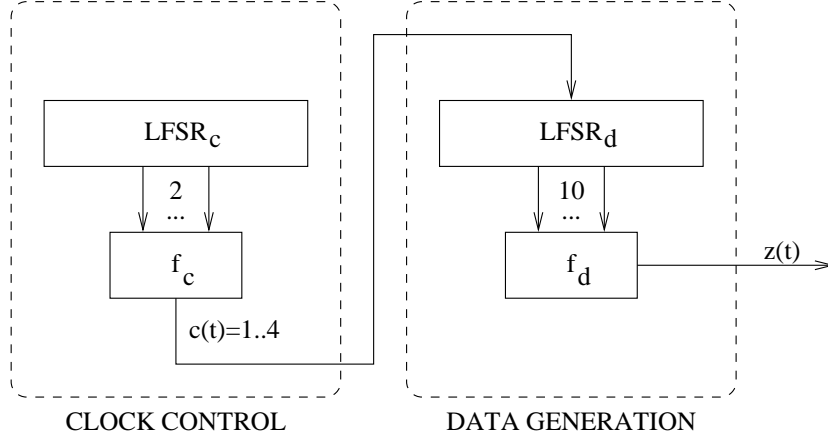
## 2 Description of LILI-128

LILI-128 uses two LFSRs,  $LFSR_c$  and  $LFSR_d$ .  $LFSR_c$  has an internal state of 39 bits and is clocked once for each output bit.  $LFSR_d$  has an internal state of 89 bits and is clocked 1 to 4 times, depending on two bits in  $LFSR_c$ . During key setup phase a  $128 = 39 + 89$  bit cryptovvariable is directly loaded into these two registers.<sup>1</sup>

In the following, we let  $t_0, t_1, \dots, t_{38}$  denote the individual bits of  $LFSR_c$ ,  $t_0$  being the most significant bit in the register and  $t_{38}$  being the least significant

---

<sup>1</sup> In [2] the authors also discuss other keying methods for LILI-128.



**Fig. 1.** Overview of the LILI-128 keystream generator.

bit. Similarly we use  $u_0, u_1, \dots, u_{88}$  to denote the individual bits of  $LFSR_d$ . The primitive polynomial for  $LFSR_c$  is

$$x^{39} + x^{35} + x^{33} + x^{31} + x^{17} + x^{15} + x^{14} + x^2 + x + 1$$

while  $LFSR_d$  uses the primitive polynomial

$$x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1.$$

The procedure for generating keystream is as follows:

1. Ten bits from  $LFSR_d$  are fed to a highly nonlinear function  $f_d, f_d : \mathbb{F}_2^{10} \rightarrow \mathbb{F}_2$  to generate one output bit  $z(t)$ .<sup>2</sup>

$$z(t) = f_d(u_0, u_1, u_3, u_7, u_{12}, u_{20}, u_{30}, u_{44}, u_{65}, u_{80}).$$

2. Two bits from  $LFSR_c$  are fed to a “linear” clock control function  $f_c : \mathbb{F}_2^2 \rightarrow \mathbb{F}_2$  to form the clocking amount  $c(t)$ :

$$c(t) = f_c(t_{12}, t_{20}) = 2t_{12} + t_{20} + 1$$

3. The clock control register  $LFSR_c$  is clocked once and the data generation register  $LFSR_d$  is clocked  $c(t)$  times (i.e. 1, 2, 3, or 4 times).

Note that the output bit is indeed generated *before* the LFSRs are clocked, hence effectively halving the key search effort in some applications.

<sup>2</sup> The  $f_d$  function is specified as a 1024-entry table in the original specification [1], and is excluded from this paper since it is irrelevant to the present attack.

**Lemma 1.** For each  $\Delta_c = 2^{39} - 1$  times  $LF\text{SR}_c$  is clocked,  $LF\text{SR}_d$  is clocked exactly  $\Delta_d = 5 * 2^{38} - 1$  times. <sup>3</sup>

*Proof.* We claim that for all cryptovariables and all values  $t$ :

$$\sum_{i=1}^{2^{39}-1} c(t+i) = \Delta_d$$

Since the polynomial of  $LF\text{SR}_c$  is primitive, it's period is  $2^{39} - 1 = \Delta_c$  and thus the internal state of the register goes through all possible values except the all zero state  $t_0 = t_1 = \dots = t_{38} = 0$ . During this cycle the control bits  $(t_{12}, t_{20})$  have value  $(0, 0)$  exactly  $2^{37} - 1$  times and values  $(0, 1)$ ,  $(1, 0)$ , and  $(1, 1)$  exactly  $2^{37}$  times, bringing the total sum to  $1 * (2^{37} - 1) + (2 + 3 + 4) * 2^{37} = 1374389534719 = \Delta_d$ .

**Lemma 2.**  $LF\text{SR}_d$  can be stepped by  $\Delta_d$  number of positions forward or backward by performing a vector-matrix multiplication with a precomputed  $89 \times 89$  bit matrix over  $GF(2)$ . The matrix can be constructed with roughly  $2^{28}$  bit operations using a binary matrix exponentiation algorithm.

*Proof.* Trivial.

This could also be achieved using a multiplication algorithm in  $GF(2^{89})$ , but for a constant  $\Delta_d$  vector-matrix multiplication actually appears to be slightly faster and functionally equivalent.

### 3 The Attack

Although many tradeoffs are possible, we will present a concrete version of the attack where we have tried to minimize the amount of keystream required. The amount of off-line and on-line computation are approximately the same.

#### 3.1 Constructing the Lookup Table

A table of  $2^{45}$  89-bit words is set up by iterating the following procedure  $2^{46}$  times. A pseudorandom 89-bit value is loaded into  $LF\text{SR}_d$  and 45 bits from  $f_d$  are sampled  $\Delta_d$  steps apart (see Lemma 2). This 45-bit vector is used as an index in the table to store the original 89-bit register value.

**Analysis.** The expected proportion of filled slots in the table is  $1 - e^{-2} = 0.8647$ . The table size is  $2^{51.48}$  bits. The computational effort required to construct the table is roughly equivalent to  $2^{48}$  DES operations.

---

<sup>3</sup> This lemma follows implicitly from Theorem 2 in [1]

### 3.2 Lookup Stage

We have  $2^{46}$  bits of keystream  $z(0), z(1), \dots, z(2^{46} - 1)$ .

1. For  $i = 0$  to  $2^{46} - 44\Delta_c - 1$  Do:
2.      $\text{idx} = z(i) \mid z(i + \Delta_c) \mid \dots \mid z(i + 44\Delta_c)$
3.     Load  $\text{Table}[\text{idx}]$  to  $LF\text{SR}_d$ .
4.     Rewind  $LF\text{SR}_d$  back  $\Delta_d \lfloor \frac{i}{\Delta_c} \rfloor$  positions.
5.     For  $j = 0$  to 127 Do:
6.         If  $(f_d(LF\text{SR}_d) \neq z(j\Delta_c + (i \bmod \Delta_c)))$  break loop.
7.         Advance  $LF\text{SR}_d$  by  $\Delta_d$  positions.
8.     If previous loop was not broken, return  $LF\text{SR}_d$ .

In line 2, a 45-bit index to the lookup table is constructed. In line 3, this index is used to fetch a 89-bit candidate value for  $LF\text{SR}_d$ . In line 4, this guess for  $LF\text{SR}_d$  is rewinded back a multiple of  $\Delta_d$  steps so that its output bit should match with  $z(i \bmod \Delta_d)$  (see Lemma 1). The loop in lines 5 to 7 compares 128 bits sampled from  $LF\text{SR}_d$   $\Delta_d$  steps apart to keystream bits sampled  $\Delta_c$  bits apart. If all 128 bits match, the correct value for  $LF\text{SR}_d$  has been found with high probability and is returned on line 8. This guess can be furthermore verified by performing an exhaustive search for the 39-bit value of  $LF\text{SR}_c$ .

**Analysis.** The probability of finding the correct value for  $LF\text{SR}_d$  at least once in the main loop is

$$1 - \left(1 - \frac{0.8647 * 2^{45}}{2^{89}}\right)^{2^{46} - 44\Delta_c} \approx 90\%.$$

The inner loop on lines 5 to 7 breaks with high probability after only few tries. The main loop runs for about  $2^{45}$  iterations (before a correct value is found). Therefore we claim that the computational effort is roughly equivalent to  $2^{48}$  DES operations.

## 4 Conclusions

We have presented a novel time-memory tradeoff attack against LILI-128 that requires  $2^{46}$  bits (8 terabytes) of keystream, and therefore is not usable in many applications. We conjecture that LILI-128 can be broken using a lookup table of  $2^{45}$  89-bit words ( $2^{51.48}$  bits) and computational effort equivalent to  $2^{48}$  DES operations.

We therefore feel that the security of LILI-128 is not as high as suggested by the designers. We do not recommend the use of this encryption algorithm for high volumes of data, or as a general-purpose standard for high security applications.

## 5 Acknowledgments

The author would like to thank Steve Babbage and Kaisa Nyberg for very helpful input.

## References

1. E. Dawson, J. Golić, W. Millan and L. Simpson. *The LILI-128 Keystream Generator* Proceedings of the Seventh Annual Workshop on Selected Areas in Cryptology - SAC '2000, to appear in Springer-Verlag LNCS, 2000.
2. E. Dawson, J. Golić, W. Millan and L. Simpson. *Response to Initial Report on LILI-128*. Submitted to Second NESSIE Workshop, 2001.
3. F. Jönsson and T. Johansson. *A Fast Correlation Attack on LILI-128*. <http://http://www.it.lth.se/thomas/papers/paper140.ps>, 2001.
4. S. Babbage. *Cryptanalysis of LILI-128*. NESSIE Public Report, <https://www.cosic.esat.kuleuven.ac.be/nessie/reports>, 2001.
5. J. White. *Initial Report on the LILI-128 Stream Cipher*. NESSIE Public Report, <https://www.cosic.esat.kuleuven.ac.be/nessie/reports>, 2001.
6. P. Flajolet and A. M. Odlyzko. *Random Mapping Statistics*. in J-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology, Proceedings of EUROCRYPT'89*, Houtalen, Belgium, April 1989, LNCS 434, pages 329-354. Springer Verlag, 1990.