Cryptanalysis of LILI-128

Steve Babbage
Vodafone Ltd, Newbury, UK

22nd January 2001

Abstract: LILI-128 is a stream cipher that was submitted to NESSIE. Strangely, the designers do not really seem to have tried to ensure that cryptanalysis is no easier than by exhaustive key search. We show that there are indeed attacks faster than exhaustive key search. We also demonstrate a related key attack which has very low complexity, and which could be of practical significance if the cipher were used in a certain rather natural way.

1. Introduction

LILI-128 is a synchronous stream cipher designed by Dawson, Clark, Golić, Millan, Penna and Simpson [5], and submitted to NESSIE. It uses a 128-bit key.

No very serious effort seems to have been made by the designers to ensure that cryptanalysis of this cipher is as hard as exhaustive search on a 128-bit key. For instance they write that:

> *… we conjecture that the complexity of divide and conquer attacks on LILI-128 is at least 2^{112} operations…. This is a conservative estimate, and the true level of security may be much higher.*

But it seems reasonable to insist that any cipher recommended by NESSIE should not be subject to any attack faster than exhaustive key search. In this note we show that there are indeed attacks faster than exhaustive key search. We also demonstrate a related key attack which has very low complexity, and which could be of practical significance if the cipher were used in a certain rather natural way.

2. Overview of LILI-128

There are two LFSRs: $LFSR_c$, which is 39 bits long, and $LFSR_d$, which is 89 bits long (so a total of 128 bits of internal state). Both have primitive feedback polynomials. For each keystream bit:

- The keystream bit is produced by applying a nonlinear function f_d to 10 of the bits in $LFSR_d$. f_d is balanced, of course; it has nonlinear order 6 and correlation immunity of degree 3. The stages from which the inputs are taken form a full positive difference set.
- $LFSR_c$ is clocked once. Two bits from $LFSR_c$ determine an integer c in the range $\{1,2,3,4\}$.
- $LFSR_d$ is clocked c times.

The keystream generator is initialised simply by loading the 128 bits of key into the registers. Keys that cause either register to be initialised with all zeroes are considered invalid.

3. Time-Memory Tradeoff Attack

The simplest observation to be made is that the size of the internal state is only 128 bits, and so there are clearly time-memory tradeoff attacks faster than exhaustive search if any significant quantity of observed keystream is available (see [1, 2]).

The usual time-memory tradeoff involves:

- a preprocessing stage in which a large dictionary is built containing many (state, 128-bit keystream sequence) pairs, sorted by keystream sequence;
- an actual attack stage, in which observed (overlapping) 128-bit keystream sequences are looked up in the dictionary; if a match is found, then with high probability the associated state was the internal state of the generator when that observed keystream sequence was produced.

The basic attack introduced by Babbage [1] has complexity[1] $T = D = N/M$ and $P = M = N/D$, where T is the time for the actual attack stage, D is the quantity of observed keystream, N is the size of the internal state space (so 2^{128} in this case), M is the amount of memory required, and P is the time for the preprocessing stage. Even if a generous 2^{40} observed keystream bits were available, the dictionary would require memory for 2^{88} records, which is clearly impractical.

Biryukov, Shamir and Wagner [2,3] introduced techniques for saving memory, allowing a more flexible tradeoff $TM^2D^2 = N^2$ (and still $P = N/D$) for any $D^2 \le T \le N$. In this case, with 2^{40} observed keystream bits and memory for 2^{36} records, the time for an actual attack is 2^{104}. The memory and observed keystream requirements are just about feasible, and the time for both stages is faster than exhaustive search (although logarithmic terms have been omitted, which in practice would push the time closer to 2^{128}). With only 2^{28} observed keystream bits and memory for 2^{36} records, T becomes 2^{128}, so there is no improvement over exhaustive search.

But it must be remembered that this tradeoff is just about finding a common item in two lists: list A of keystream sequences generated from known states, and list B of observed keystream sequences. One list is sorted into a dictionary, and then items on the other list are looked up in the dictionary. As observed in [1], it can be either of the lists that is sorted into a dictionary. So even with only 2^{28} observed keystream bits, an attack is possible with time complexity 2^{100} and memory 2^{28}: sort the observed overlapping 128-bit keystream sequences into a dictionary, then repeatedly (around 2^{100} times) try a random state, generate 128 bits of keystream from it, and look for the result in the dictionary.

It is clear that any significant quantity of consecutive keystream bits (or, more generally, regularly spaced linear combinations of keystream bits) can be used in this way for an attack that is faster than exhaustive key search. The more observed bits, the faster the attack.

4. Solving Simultaneous Linear Equations

Guess the 39 key bits used to initialise the clock control register $LFSR_c$. For the correct guess, you then know exactly how many times $LFSR_d$ has been clocked when each keystream bit is generated. Each keystream bit is thus a 6^{th} order function of ten bits, each of which is a known linear combination of the remaining 89 key bits. So each keystream bit is a known linear combination of all the possible products of up to 6 of those 89 bits.

---

[1] We ignore logarithmic terms.

There are $\sum_{i=1}^{6}\binom{89}{i} = 625173825 \approx 1{\cdot}16 \times 2^{29}$ products of up to 6 from 89 bits. So with roughly that many observed keystream bits, the problem reduces to solving simultaneous linear equations in that many variables. (We also have to reject incorrect guesses for $LFSR_c$, but that is simple — either the linear equations will be inconsistent, or else their solutions will be inconsistent when interpreted as products of secret key bits.)

Without trying to implement it, it is difficult to know exactly how long solving that many simultaneous equations would take in practice. Coppersmith and Winograd [4] have an asymptotic time complexity for matrix inversion of $O(n^{2.376})$, but with a large constant factor. Strassen's algorithm [6] has complexity $7n^{2.807} - 6n^2$, so $2^{84.8}$ in this case; the overall attack would therefore have time complexity $2^{39+84.8} = 2^{123.8}$, which is marginally less than exhaustive key search. However, just storing the coefficients of the equations would require 2^{58} bits, which is impractical. So, with today's computers and algorithms, this seems to be an academic rather than a practical attack.

A rather similar attack is considered by the designers in [5], but they suggest that roughly $2^{39}\binom{89}{6}$ keystream bits would be required. As we see above, by guessing the contents of $LFSR_c$ and then performing a linearity attack just on $LFSR_d$, we eliminate the factor 2^{39}.

5. Rekeying / Related Key Attacks

It is very common for stream ciphers to be used repeatedly with the same secret key, loaded in combination with some varying non-secret initialisation vector. There is therefore good reason to consider the effect of this rekeying — which in effect amounts to a related key attack, but with rather more justification than related key attacks tend to have against block ciphers.

The simplest way to combine a secret key and an IV is to XOR them together. If LILI-128 is rekeyed in this way, then the system can become extremely weak, as we now explain.

Suppose that:

- the 128-bit secret key is k;
- a number of successive 128-bit IVs are $v_1 \ldots v_r$;
- LILI-128 is loaded (i.e. the registers initialised) with $k \oplus v_i$;
- the corresponding keystream sequences are available to the cryptanalyst.

The attack proceeds in two phases, which we will first outline and then describe in slightly more detail:

Phase 1: Guess the 39 secret key bits used to initialise the clock control register $LFSR_c$, and quickly reject incorrect guesses, so that the correct value is known. For each IV v_i, we now know exactly how many times $LFSR_d$ has been clocked when each keystream bit is generated.

Phase 2: Compare several keystream bits produced using different IVs but when $LFSR_d$ has been clocked exactly the same number of times. Deduce the secret key components of the 10 input bits to the nonlinear function f_d at that point. Repeat several times, to obtain plenty of linear equations in the 89 secret key bits used to initialise $LFSR_d$. Solve those linear equations to obtain the secret key bits.

For the detail, we will introduce some notation. When $LFSR_d$ is initialised with just the secret key k (i.e. with IV all 0s), and then clocked t times, let the 10-bit vector representing the

inputs to the nonlinear output function f_d be k_t. When $LFSR_d$ is initialised with just v_i and then clocked t times, let the 10-bit vector representing the inputs to f_d be v_{it}.

Clearly, when $LFSR_d$ is initialised with $k \oplus v_i$ and then clocked t times, the 10-bit vector representing the inputs to f_d is $k_t \oplus v_{it}$.

Detail of phase 1

We can reject incorrect guesses for $LFSR_c$ as follows. Find a value t, and different IVs v_i and v_j, such that v_{it} and v_{jt} are equal, and for which keystream bits are in fact generated in each case when, according to our guess, $LFSR_d$ has been clocked exactly t times. (For any fixed t, the probability that a keystream bit will be generated when $LFSR_d$ has been clocked exactly t times is approximately 0.4.) If our guess is correct, the two keystream bits must be equal. If it is incorrect then they will be equal with probability roughly ½.

Roughly 39 comparisons will suffice to reject all incorrect guesses, and identify the correct one.

For this method to work, there are tradeoffs between the number r of different IVs available and the length l of each keystream sequence; the nature of the tradeoffs depends to some extent on the nature of the IVs. If different IVs are independently random, then there are roughly $(0.4)(2^{-10})\binom{r}{2}l$ pairs of keystream bits with the same values of v_{it} and v_{jt}; this formula reaches the required value of ≈39 for instance when r=32 and l=202, or when r=16 and l=832, or when r=64 and l=50. We need $l \geq 20$ to ensure that the whole of $LFSR_c$ is covered. The analysis is slightly more complex if successive IVs are related, e.g. if they are successive values of some counter.

If we guess all 39 bits of $LFSR_c$ together, and then look to reject incorrect guesses, then the complexity is slightly greater than 2^{39}. But in fact we can break the work down and guess just a couple of bits at a time. (Guess the two secret key bits contributing to the first integer $c \in \{1,2,3,4\}$; confirm or reject this guess as described above; go on to the two secret key bits contributing to the second integer c; etc etc.) So the complexity of Phase 1 is very low indeed.

Detail of phase 2

We now know exactly how many times $LFSR_d$ has been clocked when each keystream bit is generated. We proceed to determine the contents of $LFSR_d$.

For some value t, find several different IVs v_i such that in each case keystream bits are generated when $LFSR_d$ has been clocked exactly t times. Then consider all possible values for k_t. For the correct value of k_t, the observed keystream bit for IV v_i will always equal $f_d(k_t \oplus v_{it})$; for incorrect values of k_t, equality will hold with probability roughly ½. Roughly ten different IVs will suffice to reject all incorrect guesses, and determine the correct one.

Determining k_t gives us ten linear equations in the secret key bits used to initialise $LFSR_d$. Repeating 10 or 11 times will give us 100–110 equations, which should be enough to determine those 89 secret key bits (there will be some overlap between the equations since the same register bit will appear repeatedly in different positions).

The complexity of Phase 2 is again extremely low. If even very short (not much more than 10-bit) keystream sequences are available for 25 different IVs, then for enough values of t the expected number of times a keystream bit is generated when $LFSR_d$ has been clocked exactly t times is approximately 10, which is sufficient. Slightly fewer keystream sequences will suffice if they are longer (the values of t with ten or more "hits" will be more scattered).

Other comments and summary

There are variations on the above process. k_0 can be determined without knowing anything at all about $LFSR_c$. Phases 1 and 2 could be combined: guess the first few bits used from $LFSR_c$, and one of k_1, k_2, k_3 and k_4, rejecting inconsistent guesses for both together; determine the rest of k_1, k_2, k_3 and k_4; go on to the next few bits used from $LFSR_c$, and so on. And we don't necessarily have to reject all but one possibility at every stage — we can keep a few possibilities "live" at once, as long as the number doesn't keep growing. With this combined approach it suffices to have roughly 30 sequences of a little over 20 bits each. Anyway, it is clear that an attack of this kind can be performed if a rather small number of rather short keystream sequences are available. And the complexity of the attack is very low (real-time, even, as far as that makes sense for an attack on multiple uses of the same cipher).

We have restricted ourselves to keys related by XORing different known IVs. If the IVs are *chosen* by the cryptanalyst — which is an optimistic but not completely fanciful assumption — then variations become possible with even smaller data requirements. Keys with more general chosen relationships would open up a host of other possibilities, but of less practical significance.

6. Design Criteria for the Nonlinear Function

The keystream bit is computed using a non-linear function on 10 of the bits in $LFSR_d$. Amongst other criteria, the function was chosen to have fairly high order correlation immunity (order 3). This choice was made to give resistance against correlation attacks.

This seems to be a misguided application of correlation immunity. Having no correlation to subsets of up to three of the input bits is rather pointless, because there is correlation to sums of four or more bits — and any sum of the bits from four or more stages of an LFSR is itself a linear sequence from the same LFSR. When all input bits come from one LFSR, sums of small numbers of input bits are no more in need of protection from correlation attacks than sums of large numbers of input bits.

As noted in section 4.3 of [5], there is merit in having at least first order correlation immunity, to prevent attacks that track a bit from one position in $LFSR_d$ to another. But correlation immunity of order greater than one seems an inappropriate criterion (the input stages to f_d form a full positive difference set, so no two bits appear together twice as inputs). A more appropriate criterion might have been to choose a balanced, first order correlation immune function with minimum correlation to any linear function of more than one bit. (The present author does not know whether any such function does actually achieve better nonlinearity than the LILI-128 function — this observation is about the criteria for selecting the function rather than the function itself.)

7. Conclusions

General: If a general-purpose cipher has a 128-bit key, it is expected that there should be no attack faster than 128-bit exhaustive search. But it does not appear that the designers of LILI-128 have really tried to ensure that there are no attacks faster than exhaustive key search; there are various faster attacks, including at least one very straightforward one.

Related key attacks: for better or for worse, related key attacks against block ciphers are taken seriously. A related key attack faster than exhaustive key search against one of the AES candidates would have been enough to remove it from contention. We have demonstrated a related key attack against LILI-128 which requires only a few tens of related keys, and has very low complexity; we have also shown how those conditions could be available in

practical use if the system is rekeyed in a certain very natural way. Even if it is specified that LILI-128 should be rekeyed in a different way, the general concept of the related key attack remains; we leave it to others to decide whether that matters.

8. References

[1] S.H.Babbage, *Improved "Exhaustive Search" Attacks on Stream Ciphers*, ECOS 95 (European Convention on Security and Detection), IEE Conference Publication No. 408, May 1995.

[2] A.Biryukov, A.Shamir, *Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers*, ASIACRYPT 2000, published as LNCS 1976, Springer Verlag, 2000.

[3] A.Biryukov, A.Shamir, D.Wagner, *Real Time Cryptanalysis of A5/1 on a PC*, FSE 2000, to be published in LNCS series by Springer Verlag.

[4] D.Coppersmith, S.Winograd, *Matrix multiplication via arithmetic progressions*, in Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, New York City, 25-27 May 1987.

[5] E.Dawson, A.Clark, J.Golić, W.Millan, L.Penna, L.Simpson, *The LILI-128 Keystream Generator*, NESSIE submission, in the proceedings of the First Open NESSIE Workshop (Leuven, November 2000), and available at http://www.cryptonessie.org

[6] V.Strassen, *Gaussian Elimination is Not Optimal*, Numerische Mathematik, vol 13, pp 354-356, 1969.