A New 128-bit Key Stream Cipher LEX

Alex Biryukov

Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC,
Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium
`http://www.esat.kuleuven.ac.be/~abiryuko/`

Abstract. A proposal for a simple AES-based stream cipher which is at least 2.5 times faster than AES both in software and in hardware.

1 Introduction

In this paper we suggest a simple notion of a *leak extraction* from a block cipher. The idea is to extract parts of the internal state at certain rounds and give them as the output key stream (possibly after passing an additional filter function). This idea applies to any block cipher but a careful study by cryptanalyst is required in each particular case in order to decide which parts of the internal state may be given as output and at what frequency. This mainly depends on the strength of the cipher's round function and on the strength of the cipher's key-schedule. For example, ciphers with good diffusion might allow to output larger parts of the internal state at each round than ciphers with weak diffusion.

2 Description of LEX

In this section we describe a 128-bit key stream cipher LEX (which stands for Leak EXtraction, and is pronounced "leks"). The design is simple and is using AES in a natural way: at each AES round output certain four bytes from the intermediate variable. The AES with all three different key lengths (128, 192, 256) can be used. The difference with AES is that the attacker never sees the full 128-bit ciphertext but only portions of the intermediate state. Similar principle can be applied to any other block-cipher.

In Fig. 1 we show how the cipher is initialized and chained. First a standard AES key-schedule for some secret 128-bit key K is performed. Then a given 128-bit IV is encrypted by a single AES invocation: $S = AES_K(IV)$. The 128-bit result S together with the secret key K constitute a 256-bit secret state of the stream cipher.[1] S is changed by a round function of AES every round and K is kept unchanged (or in a more secure variant is changing every 500 AES encryptions).

The most crucial part of this design is the exact locations of the four bytes of the internal state that are given as output as well as the frequency of outputs

[1] In fact the K part is expanded by the key-schedule into ten 128-bit subkeys.

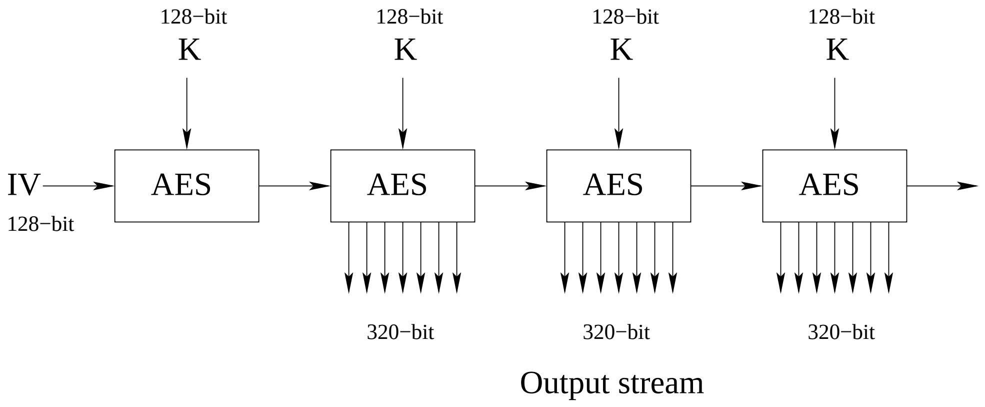

Fig. 1. Initialization and stream generation.

(every round, every second round, etc.). So far we suggest to use the bytes $b_{0,0}, b_{2,0}, b_{0,2}, b_{2,2}$ at every odd round and the bytes $b_{0,1}, b_{2,1}, b_{0,3}, b_{2,3}$ at every even round. We note that the order of bytes is not relevant for security but is relevant for fast software implementation. The order of bytes as given above allows to extract a 32-bit value from two 32-bit row variables t_0, t_2 in just four steps (that can be pipelined):

$$out32 = ((t_0 \& 0xFF00FF) << 8) \oplus (t_2 \& 0xFF00FF),$$

while each round of AES uses about 40 steps. Here t_i is a row of four bytes: $t_i = (b_{i,0}, b_{i,1}, b_{i,2}, b_{i,3})$. So far we do not propose to use any filter function and output the bytes as they are. The choice of the output byte locations (see also

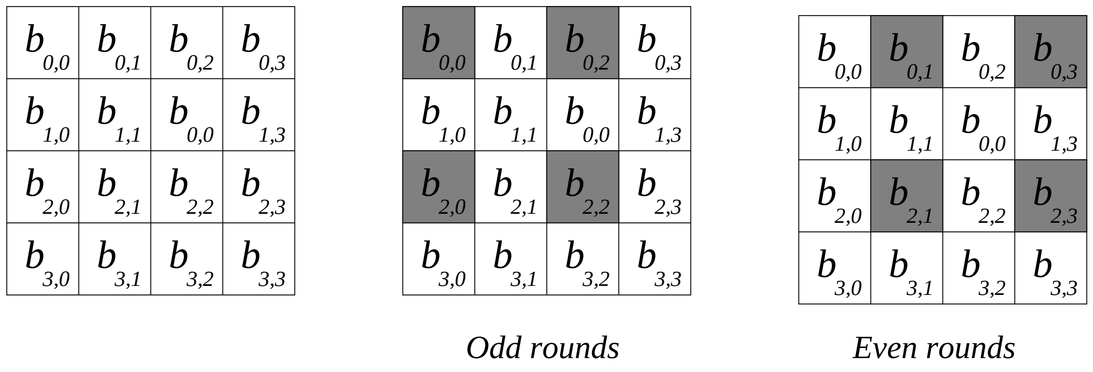

Fig. 2. The positions of the leak in the even and in the odd rounds.

Fig. 2) is motivated by the following: both sets constitute an invariant subset of the `ShiftRows` operation (the first row is not shifted and the third is rotated by two bytes). By alternating the two subsets in even and odd rounds we ensure that the attacker does not see input and output bytes that are related by a single `SubBytes` and a single `MixColumn`. This choice ensures that the attacker will have to analyze two consecutive rounds. The two rounds of AES have full diffusion thus limiting divide-and-conquer capabilities of the attacker.

The speed of this cipher is exactly 2.5 times faster than AES up to a cost of extraction of four bytes at each round.

3 Analysis of LEX

In this section we analyze resistance of LEX to various attacks.

3.1 Period of the Output Sequence

The way we use AES is essentially an Output Feedback Mode (OFB), in which instead of using the ciphertexts as a key-stream we use the leaks from the intermediate rounds as a key-stream. The output stream will eventually cycle when we traverse the full cycle of the AES-generated permutation. If one assumes that AES is indistinguishable from a random permutation for any fixed key, one would expect the cycle size to be of the order $O(2^{128})$ since the probability of falling into one of the short cycles is negligible[2].

If the output stream is produced by following a cycle of a random permutation it is easily distinguished from random after observing absence of 128-bit collisions in a stream of 2^{64} outputs. In our case since we output only part of the state at each round, the mapping from internal state to the output is not 1-1 and thus such collisions would occur.

3.2 Tradeoff Attacks

For a stream cipher to be secure against time-memory and time-memory-data tradeoff attacks [1, 5, 2] the following conditions are necessary: $|K| = |IV| = |State|/2$. This ensures that the best tradeoff attack has complexity roughly the same as the exhaustive key-search. The IV's may be public, but it is very important that full-entropy IV's are used to avoid tradeoff-resynchronization attacks [6]. In the case of LEX $|K| = |IV| = |Block| = 128$ bits, where $Block$ denotes an intermediate state of the plaintext block during the encryption. Internal state is the pair (IV, K) at the start and $(Block, Key)$ during the stream generation, and thus $|K| + |IV| = |K| + |S| = 256$ bits which is enough to avoid the tradeoff attacks. Note that if one uses LEX construction with larger key variants of AES this might be a problem. For example for 192-bit key AES the state would consist of 128-bit internal variable and the 192-bit key. This would

[2] A random permutation over n-bit integers typically consists of only about n cycles, the largest of them spanning about 62% of the space.

allow to apply a time-memory-data tradeoff attack with roughly 2^{160} stream, memory and time. Such attack is absolutely impractical but may be viewed as a certificational weakness.

3.3 Algebraic Attacks

Algebraic attacks on stream ciphers [4] are a recent and a very powerful type of attack. Applicability of these to LEX is to be carefully investigated. If one could write a non-linear equation in terms of the outputs and the key – that could lead to an attack. Re-keying every 500 AES encryptions may help to avoid such attacks by limiting the number of samples the attacker might obtain while targeting a specific subkey. We expect that after the re-keying the system of non-linear equations collected by the attacker would become obsolete. Shifting from AES key-schedule to a more robust one might be another precaution against these attacks. Note also that unlike in LFSR-based stream ciphers we expect that there do not exist simple relations that connect internal variables at distances of 10 or more steps. Such relations if they would exist would be useful in cryptanalysis of AES itself.

3.4 Differential, Linear or Multiset Resynchronization Attacks

If mixing of IV and the key is weak the cipher might be prone to chosen or known IV attacks similar to the chosen plaintext attacks on the block-ciphers. However in our case this mixing is performed via a single AES encryption. Since AES is designed to withstand such differential, linear or multiset attacks we believe that such attacks pose no problem for our scheme either. Slide-resynchronization attacks [3] should also not be a problem due to different round constants in the key-schedule which break the self-similarity of the AES.

3.5 Potential Weakness – AES Key-schedule

There is a simple way to overcome weaknesses in AES key-schedule (which is almost linear) and which might be crucial for our construction. The idea might be to use ten consecutive encryptions of the IV as subkeys, prior to starting the encryption. This method will however loose in key agility, since key-schedule time will be 11 AES encryptions instead of one. If better key-agility is required a faster dedicated key-schedule may be designed.

If bulk encryption is required then it might be advisable to replace static key by slowly time-varying key. One possibility would be to perform additional 10 AES encryptions every 500 AES encryptions and to use the 10 results as subkeys. This method quite efficient in software might not be suitable for small hardware (Profile 2) due to the requirement to store 1280 bits (160 bytes) of the subkeys. The overhead of such key-change is only 2% slowdown, while it might stop potential attacks which require more than 500 samples gathered for a specific subkey. This method quite efficient in software might not be suitable for small hardware (Profile 2) due to the requirement to store 1280 bits (160 bytes) of the subkeys. An alternative more gate-efficient solution would be to perform single AES encryption every 100 steps without revealing the intermediate values and use the result as a new 128-bit key. Then use the keyschedule of AES to generate the subkeys. Note, that previously by iterating AES with the same key we explored a single cycle of AES, which was likely to be of length $O(2^{128})$ due to the cipher being a permutation of 2^{128} values. However by doing intermediate key-changes we are now in a random mapping scenario. Since state size of our random mapping is 256 bits (key + internal state), one would expect to get into a "short cycle" in about $O(2^{128})$ steps, which is the same as in the previous case and poses no security problem.

3.6 No Weak Keys

Since there are no weak keys known for the underlying AES cipher we believe that weak keys pose no problem for this design either. This is especially important since we suggest frequent rekeying to make the design more robust against other cryptanalytic attacks.

3.7 Dedicated Attacks

An obvious line of attack would be to concentrate on every 10th round, since it reuses the same subkey, and thus if the attacker guesses parts of this subkey he still can reuse this information $10t, t = 1, 2, \ldots$ rounds later. Note however that unlike in LFSR or LFSM based stream ciphers the other parts of the intermediate state have hopelessly changed in a complex non-linear manner and any guesses spent for those are wasted (unless there is some weakness in a full 10-round AES).

4 Implementation

We believe that LEX would be able to run about 2.5 times faster than AES in both hardware and software. Since LEX could reuse existing AES implementations it might provide a simple and cheap speedup option in addition to the already existing base AES encryption. For example, if one uses a fast software AES implementation which runs at 14-15 clocks per byte we may expect LEX to be running at about 5-6 clocks per byte. The same leak extraction principle naturally applies to 192 and 256-bit AES resulting in LEX-192 and LEX-256. LEX-192 should be 3 times faster than AES-192, and LEX-256 is 3.5 times faster than AES-256. Note that unlike in AES the speed penalty for using larger key versions is much smaller in LEX (a slight slowdown for a longer keyschedule, i.e. resynchronization but not for the stream generation).

5 Strong Points of the Design

Here we list some benefits of using this design:

- AES hardware/software implementations can be reused with few simple modifications. The implementors may use all their favorite AES implementation tricks.
- The cipher is at least 2.5 times faster than AES. In order to get an idea of the speed of LEX divide performance figures of AES by a factor 2.5. The speed of key and IV setup is equal to the speed of AES keyschedule followed by a single AES encryption. In hardware the area and gate count figures are essentially those of the AES.
- Unlike in the AES the key-setup for encryption and decryption in LEX are the same.
- The cipher may be used as a speedup alternative to the existing AES implementation and with only minor changes to the existing software or hardware.
- Security analysis benefits from existing literature on AES.
- The speed/cost ratio of the design is even better than for the AES and thus it makes this design attractive for both fast software and fast hardware implementations (Profile 1). The design will also perform reasonably well in restricted resource environments (Profile 2).
- Since this design comes with explicit specification of IV size and resynchronization mechanism it is secure against time-memory-data tradeoff attacks. This is not the case for the AES in ECB mode or for the AES with IV's shorter than 128-bits.
- Side-channel attack countermeasures developed for the AES will be useful for this design as well.

6 Summary

In this paper we have described efficient extensions of AES into the world of stream ciphers. We expect that (if no serious weaknesses of this approach would be found) it may provide a very useful speed up option to the existing base implementations of AES. We hope that there are no attacks on this design faster than $O(2^{128})$ steps. The design is rather bold and of course requires further study. However so far there are no weaknesses known to the designers as well as there are no hidden weaknesses inserted by the designers.

7 Acknowledgement

This paper is a result of several inspiring discussions with Adi Shamir. We would also like to thank Christophe De Cannière, Joseph Lano, Ingrid Verbauwhede and other cosix for the exchange of views on the stream cipher design.

References

[1] S. Babbage, "Improved "exhaustive search" attacks on stream ciphers," in *ECOS 95 (European Convention on Security and Detection)*, no. 408 in IEE Conference Publication, May 1995.

[2] A. Biryukov and A. Shamir, "Cryptanalytic time/memory/data tradeoffs for stream ciphers," in *Proceedings of Asiacrypt'00* (T. Okamoto, ed.), no. 1976 in Lecture Notes in Computer Science, pp. 1–13, Springer-Verlag, 2000.

[3] A. Biryukov and D. Wagner, "Slide attacks," in *Proceedings of Fast Software Encryption – FSE'99* (L. R. Knudsen, ed.), no. 1636 in Lecture Notes in Computer Science, pp. 245–259, Springer-Verlag, 1999.

[4] N. T. Courtois and W. Meier, "Algebraic attacks on stream ciphers with linear feedback," in *Advances in Cryptology – EUROCRYPT 2003* (E. Biham, ed.), Lecture Notes in Computer Science, pp. 345–359, Springer-Verlag, 2003.

[5] J. D. Golic, "Cryptanalysis of alleged A5 stream cipher," in *Advances in Cryptology – EUROCRYPT'97* (W. Fumy, ed.), vol. 1233 of *Lecture Notes in Computer Science*, pp. 239–255, Springer-Verlag, 1997.

[6] J. Hong and P. Sarkar, "Rediscovery of time memory tradeoffs," 2005. `http://eprint.iacr.org/2005/090`.