The LOCAL attack: Cryptanalysis of the authenticated encryption scheme ALE

Dmitry Khovratovich and Christian Rechberger

[1] University of Luxembourg, dmitry.khovratovich@uni.lu
[2] DTU, Denmark, crec@dtu.dk

Abstract. We show how to produce a forged (ciphertext,tag) pair for the scheme ALE with data and time complexity of 2^{102} ALE encryptions of short messages and the same number of authentication attempts. We use a differential attack based on a local collision, which exploits the availability of extracted state bytes to the adversary. Our approach allows for a time-data complexity tradeoff, with an extreme case of a forgery produced after 2^{119} attempts and based on a single authenticated message. Our attack is further turned into a state recovery and a universal forgery attack with a time complexity of 2^{120} verification attempts using only a single authenticated 48-byte message.

1 Introduction

Cryptanalysis and design of authenticated encryption primitives are getting renewed interest, not least because of the CEASAR initiative [1]. Recently, at DIAC 2012 and FSE 2013, a proposal named ALE was presented by Bogdanov et. al [6]. ALE provides online single-pass encryption and authentication functionality with optional processing of associated data in a single primitive. The design borrows well tested ideas from Pelican-MAC [8] and the AES-based stream-cipher LEX [3]. From an implementation point of view it is an attractive proposal as it both lends itself to lightweight hardware implementation, and at the same time offers very high speed software implementations on platforms with AES instructions available.

The designers claim 128-bit security against state recovery, key recovery, or forgery attacks, under the assumptions that nonces are not re-used. Our cryptanalysis suggests that the security against forgery and state recovery attacks is less than expected and claimed. Even though the designers limited the amount of data that can be authenticated or both authenticated and encrypted to 2^{45} bytes, our forgery attack will likely succeed. In fact, for a variant of our approach, as little as 32 bytes of available data are enough. Forthermore our approach can be extended to recover the full 256-bit internal state of ALE.

Our methods. We use differential cryptanalysis despite the designers' intention of making these attacks unlikely. Their motivation comes from the good properties of the AES round function when iterated a few times, leading to very low bounds

for the probability of differential characteristics and differentials. Study of so-called extinguishing differentials in the context of Pelican-MAC backs up this analysis.

Our attack uses differentials of a particular type, called "local collisions", as they lead to the same tags for different plaintexts. These seem to have been first used in the collision search of SHA-0 [7], and more recently in related-key key-recovery attacks on AES-192 and AES-256 [4], and are also related to the aforementioned extinguishing differentials from the security analysis of Pelican-MAC [8]. However, as we discovered, using information that is leaked via the ciphertext these local collisions can be constructed much faster than expected, in turn leading to forgery attacks. Because of these properties, we call our method the **LOCAL** method:"LOcal-Collision Amplification via Leakage".

Outline of the paper and our results. We give a short introduction into the state of the art in the authenticated encryption in Section 2. We also provide a detailed description of ALE and discuss its similarities and differences to LEX. Then we proceed with the description of our attack in Section 3. We show that each encrypted message has many counterparts which yield the same tag with probability from 2^{-119} to 2^{-102}. Hence we can use a time-data tradeoff and demonstrate the fastest attack when 2^{102} messages are available, and the slowest with complexity 2^{119} when only a single message is available. In Section 4 we turn this attack into a stronger attack, allowing for state recovery and hence universal forgery. We discuss various repair strategies in Section 5 and conclude that a version of ALE resistant to our attack would have to suffer about 30% in performance.

2 Authenticated encryption schemes and ALE

It has been known for a while that the encryption modes CBC, CFB, and CTR do not provide any sort of data integrity. Whenever a recipient of a ciphertext needs to check whether it was not modified by an adversary, a separate mechanism is needed. A traditional way to authenticate the ciphertext is to compute a message authentication code (MAC) of it, also called a tag. A secure way to do it, known as Encrypt-then-MAC, is to produce a MAC on another key and couple it with the ciphertext. A combination of a secure mode of operation and a secure MAC yields a secure authenticated encryption scheme [2], which provides

- Confidentiality (inability to distinguish the ciphertext from a random string);
- Ciphertext integrity (inability to find a valid pair (ciphertext, tag)).

Apart from using two different constructions, this approach has one clear disadvantage: it uses two different keys, which puts additional burden on the end user.

Since at least the year of 2000, cryptographers have tried to design an authenticated encryption scheme, which would use a single key and would be at least as efficient as Encrypt-then-MAC. The research went in two directions. The first one deals with new modes of operation which use an arbitrary block cipher. The ISO standards [13] GCM, CCM, and OCB are typical examples. The patented OCB mode runs almost as fast as the counter encryption mode, which yields the speed below one cycle per byte on modern CPUs if used with AES [11]. The second approach deals with dedicated AE schemes, such as Nessie submissions like Helix or Sober-128, the eStream candidate Phelix, or Grain128a. Both approaches typically use probabilistic encryption to achieve confidentiality, and nonces are the usual source of randomness.

Modern authenticated encryption schemes are also able to authenticate so called *associated data* (AD) without encrypting it [12]. A typical application is Internet packets, whose contents are encrypted, whereas headers are not for routing purposes, while they still should be bound to the encrypted data.

Syntax of authenticated encryption. It is customary to use the following syntax for a nonce-based authenticated encryption scheme with associated data. The encryption function $\mathcal{E}$ operates as follows:

$$\mathcal{E} : \mathcal{K} \times \mathcal{M} \times \mathcal{N} \times \mathcal{A} \longrightarrow \mathcal{C},$$

where $\mathcal{K}$ is the key space, $\mathcal{M}$ is the message (plaintext) space, $\mathcal{N}$ is the nonce space, $\mathcal{A}$ is the associated data space, and $\mathcal{C}$ is the ciphertext space. The authentication part of the ciphertext may be syntactically separated and called a tag $T \in \mathcal{T}$.

The decryption function decrypts valid ciphertexts into plaintexts, and invalid ciphertexts into an error ($\perp$):

$$\mathcal{D} : \mathcal{K} \times \mathcal{C} \times \mathcal{N} \times \mathcal{A} \longrightarrow \mathcal{M} \cup \{\perp\}.$$

Security against forgery attacks comes from the inability of the computationally bounded adversary to produce a ciphertext that does not decrypt to $\perp$.

Attack model. Though particular applications may have their own restrictions, the security of the authenticated encryption scheme is defined with respect to a quite powerful adversary [12]. She may ask almost arbitrary requests to encryption and decryption oracles, with the main restriction that nonces do not repeat in encryption requests (so called nonce-respective adversary). Usually, no security is offered if the sender reuses the nonce. However, the receiver usually does not have technical means to check whether the nonce has not been used in another communication. Hence an adversary may ask to decrypt several tuples (C, N, A) with the same nonce (authenticating herself to distinct receivers if needed). A secure authenticated encryption scheme returns $\perp$ even in this case.

It is said that the adversary can create a *forgery*, if she is able to submit a tuple (C, N, A) to the decryption oracle such that

– It does not return $\perp$.
– There have been no encryption request which contained N and A and returned C.

This definition does not specify whether the adversary can choose the message she wants to be authenticated. From the practical point of view, we say that the adversary constructs a *universal forgery* if she indeed can choose the message at her own, and an *existential forgery* if she can not.

Description of ALE

The authenticated encryption scheme ALE [6] is a dedicated scheme, which uses components of the AES-128 block cipher.

AES. AES-128 operates on a 16-byte block, which is traditionally represented as a matrix:

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Plaintext of AES-128 undergoes a sequence of 10 rounds, each preceded with a subkey addition. One round consists of the following invertible transformations:

– SubBytes (SB) — nonlinear bytewise transformation. Each byte enters a so called S-box (the same for the whole cipher). S-box has a maximal differential probability of 2^{-6} (four conforming inputs), but the majority of differentials have probability of either 2^{-7} or zero;
– ShiftRows (SR) — rotates row i in the array (counting from 0) by i positions to the left;
– MixColumns (MC) — linear columnwise transformation. Invertible, has branch number 5, i.e. two inputs differing in k bytes have outputs differing in at least $5 - k$ bytes, and vice versa.

The last tenth round lacks MixColumns but is followed by another subkey addition.

The key schedule of AES-128 is a lightweight transformation that produce subkeys in an invertible way.

ALE. ALE encrypts plaintexts up to 2^{45} bytes long. The nonces and keys are 128-bit strings. The encryption proceeds as follows (Figure 1). During *the initialization phase* the 128-bit none N is encrypted on the 128-bit master key K to produce the temporary key K_1. The zero 128-bit string is encrypted on K to produce the temporary state S_1. The state S_1 is then encrypted on K_1 with 10 AES rounds. The last subkey of the latter encryption is denoted by K_2.

The associated data is appropriately padded and split into 16-byte blocks. The *associated data phase* alternates injecting the AD blocks into the state with

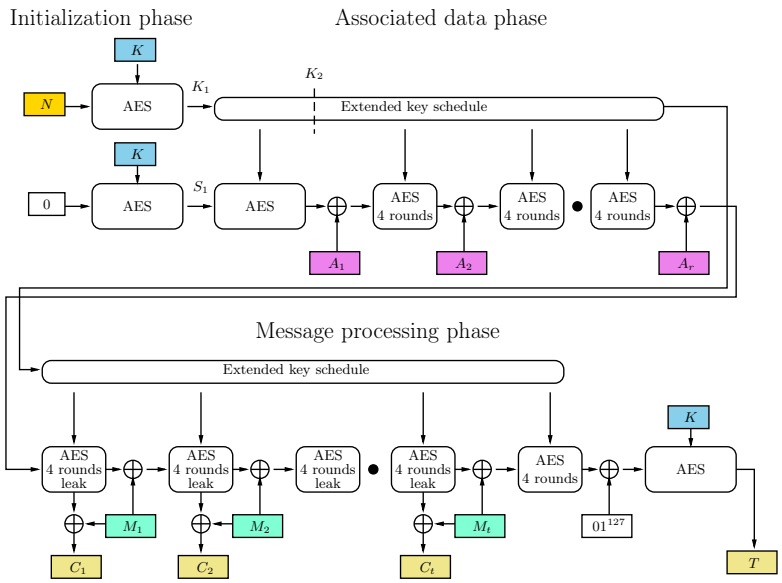

Fig. 1. Outline of authenticated encryption scheme ALE for messages multiple of block length.

encrypting the state with 4 AES rounds. The AD blocks are 16 bytes long and are simply xored into the internal state. The encryption subkeys are taken from the AES key schedule algorithm applied to K_2 and extended for as many rounds as needed (the original paper is a bit vague on the details, and we'll return to this issue in Section 4). This process continues in the message processing phase.

The message is partitioned into 16-byte blocks. For the sake of simplicity, we consider only the case where the message byte length is a multiple of 16. Then the *message processing phase* alternates groups of four leaking rounds with message block injections. Every odd round the scheme extracts bytes 0, 2, 8, 10, and every even round it extracts bytes 4, 6, 12, 14. The bytes are extracted after the SubBytes operation.

A message block is xored to the internal state and is simultaneously xored to the last 16 bytes extracted, which forms a new block of ciphertext C. After the full message is processed, the scheme encrypts the state with four rounds using the previous subkeys, xors `0x70` to byte 0, and encrypts the state again with the key K for the full 10 rounds of AES-128. The result is declared the authentication tag T.

Security claims. ALE designers claim the following: *"Any forgery attack not involving key recovery/internal state recovery has a success probability at most* 2^{-128}*"*.

Differences between LEX and ALE and design weaknesses. ALE inherited a lot from the stream cipher LEX [3], which generates the keystream also by outputting specific bytes of the AES internal state. There are two crucial differences between them apart of the authentication option: first, LEX uses the same key in all his 10-group rounds, and second, LEX does not feed any data to the internal state. The former property led to distinguishing attacks on LEX based on colliding states [9]. Distinct keys in ALE make these attacks irrelevant.

However, the latter difference actually weakens the design, as the attacker is now able to manipulate the internal state, whose contents he has just observed via leakage. Even though the extracted bytes and the message injections are separated by subkey additions, the differential analysis bypasses this countermeasure, as we see below.

3 Forgery Attack

Outline. In this section we demonstrate a forgery attack on ALE. Our goal is to produce a fresh tuple (C, N, A) that does not decrypt to $\perp$ (here C includes the tag T). An adversary first asks for the encryption of some messages, and then attempts to forge the tag by modifying ciphertexts. Even though nonces repeat in forgery attempts, they do not repeat in encryption requests. Therefore, our attack operates in a standard model.

Attack overview. The attack proceeds as follows. We ask for the encryption of a message $M = (M_1, M_2)$:

$$\mathcal{E}_K(M, N, A) = C.$$

We do not care about the message contents, the nonce, and the associated data, so the attack can be entirely *known-plaintext* as long as the plaintexts are at least two blocks long. Then we attempt to construct a pair of differences $\Delta = (\Delta_1, \Delta_2)$, which yields a *local collision* in ALE if being applied to (M_1, M_2), meaning that the two differences compensate each other. If the local collision property holds, the authentication tag remains the same, and ciphertext is simply xored with $\Delta_C = (\Delta_1, \Delta_2, 0^{128})$:

$$\mathcal{D}_K(C \oplus \Delta_C, N, A) = M.$$

If it does not hold, we repeat the procedure for another difference or another message, as explained below.

The designers of ALE supposedly ruled out such an attack, since the group of four rounds of AES between the message injection benefits from the wide trail strategy. The latter concept enables to prove that any 4-round differential trail activates at least 25 S-boxes, which yields the maximum probability of $2^{-25 \cdot 6} = 2^{-150}$. It should make any differential event, including the local collision, highly unlikely. However, this idea does not take into account the fact that as many as 16 bytes from the internal states have been extracted during these four rounds. Since they are known to the adversary, he can select the differential trail so that it has higher probability than the wide trail strategy offers. A differential trail is easily converted to a verification attempt.

Attack details. First we note that the extracted bytes are the S-box outputs (the inputs would work too). Hence whenever a trail activates an S-box whose value is extracted, the difference propagation is deterministic in this S-box, and it does not add a factor to the total probability. Thus we attempt to find a trail that has low weight and this weight consists of as many "extraction" S-boxes as possible.

We did not do an exhaustive search for all low-weight trails, but the following round weights are good enough for our purposes:

$$16 \xrightarrow{SR,MC} 4 \xrightarrow{SR,MC} 1 \xrightarrow{SR,MC} 4 \xrightarrow{SR,MC} 16.$$

The optimal layout for active S-boxes is to be determined, but the one at Figure 2 is good enough, as only 17 active S-boxes out of 25 add a factor to the probability.

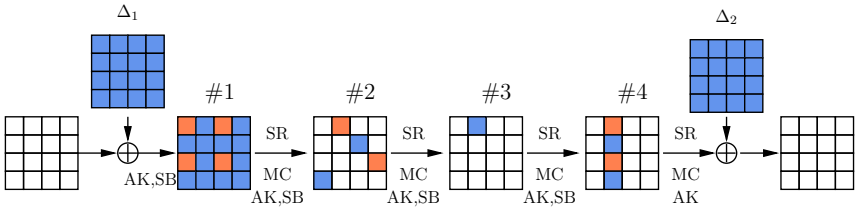

Fig. 2. Differential trail for a local collision: overview. Orange cells are active extraction S-boxes, violet cells are the other active S-boxes.

These trails can be constructed online very quickly in the start-from-the-middle framework. We select a random difference in state #3 and expand it in both directions. Whenever we encounter extraction S-boxes or MixColumns, the difference evolves deterministically. For each active non-extraction S-box we select an output difference so that the differential probability equals the maximum 2^{-6}. Eventually we obtain values of Δ_1 and Δ_2. Hence for every extraction tuple it is easy to obtain a differential trail that holds with probability $2^{-17 \cdot 6} = 2^{-102}$.

Therefore, for each encrypted 2-block message we can construct a counterpart that yields the same authentication tag with probability 2^{-102}. Hence we can construct a forgery for ALE with complexity of 2^{102} ALE encryptions of two-block messages and 2^{102} verification attempts. While it is enough to constitute a weakness in ALE, the data complexity should be reduced further to match the design restrictions.

Reducing the data complexity. The specification [6] requires that no more than 2^{40} 2-block messages be authenticated with a single key. In order to match this condition, we use a simple tradeoff by allowing some $r \leq 17$ S-boxes in a trail to have non-maximal differential probability. Instead of one choice per S-box,

we now have 2^7 choices per non-optimal S-box, and hence many more trails for the same message. The value $r = 8$ yields $\binom{17}{8}2^{56} \approx 2^{70.5}$ trails with probability 2^{-110}. Hence we can use 2^{40} plaintexts to generate $2^{110.5}$ verification attempts with the total attack probability close to 1. By further increasing r we can work with very low data complexity up to the extreme case of one message block, where we have to use all the degrees of freedom in each S-box so that the attack complexity increases to $2^{7 \cdot 17} = 2^{119}$.

The memory complexity of our attack is negligible, as we store only several AES internal states and the S-box difference distribution table.

4 Turning the forgery into a state recovery attack

The fact that the forgery from above is the result of a differential attack reveals much information about the internal state. Indeed, as long as the differential trail holds, each active S-boxes takes at most 4 possible values (2 if the probability is 2^{-7}). Hence we obtain at least 116 bits of information about the state #1. This may seem insufficient to fully recover the state and the key, as they take 256 bits altogether.

However, we note that the local collision attack can be repeated for the same message but another pair of blocks (Figure 3). Assume that we have mounted the forgery attack with local a collision based on blocks (M_2, M_3), whereas the first block is M_1. Then we attempt to construct another local collision based on blocks (M_1, M_2) with a trail of the following form (again, SB and AK are omitted):

$$4 \xrightarrow{SR,MC} 1 \xrightarrow{SR,MC} 4 \xrightarrow{SR,MC} 16 \xrightarrow{SR,MC} 16$$

As soon as we construct the second forged ciphertext, we obtain information about the internal state in the last round where all S-boxes are active. Having 4 S-boxes extracted, we obtain $12 \cdot 7 + 4 \cdot 8 = 116$ bits of information — the same as for the first local collision. Let us guess the unknown 12 bits in both fully active states and recompute the states towards the injection of M_2. Let us denote the subkeys encompassing the injection of M_2 by K_a and K_b. Then we obtain the following equation:

$$C_0 \oplus K_a \oplus M_2 \oplus K_b = C_1,$$

where C_0 and C_1 are known constants. Hence we obtain the value $K_a \oplus K_b$.

The original specification says that K_b is derived from K_b by applying an AES key schedule round with a specific constant:

$$\begin{aligned} K_b[0 \dots 3] &= F(K_a[12 \dots 15]) \oplus K_a[0 \dots 3]; \\ K_b[4 \dots 7] &= K_b[0 \dots 3] \oplus K_a[4 \dots 7]; \\ K_b[8 \dots 11] &= K_b[4 \dots 7] \oplus K_a[8 \dots 11]; \\ K_b[12 \dots 15] &= K_b[8 \dots 11] \oplus K_a[12 \dots 15]. \end{aligned}$$

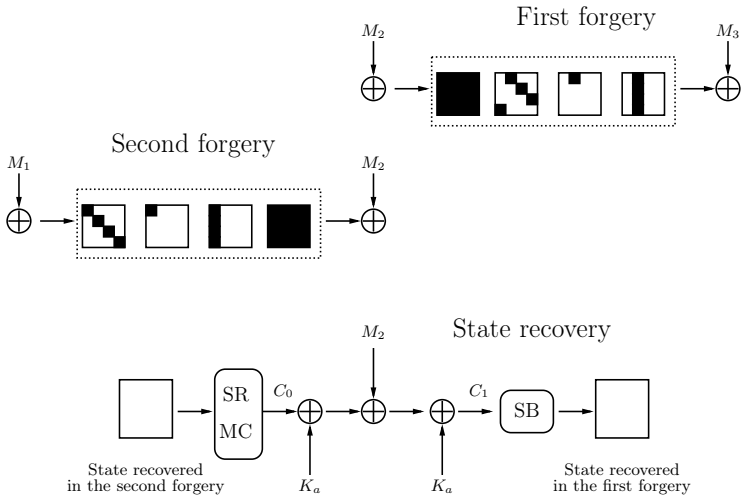

Fig. 3. Outline of the state recovery attack on ALE.

where F is an invertible nonlinear function, and $K[x..y]$ is a tuple of the key state bytes from x till y included.

It is easy to see that we can derive $K_b[0\ldots11]$ and $F(K_a[12\ldots15])$ from $K_a \oplus K_b$, which easily yields the full K_b. Since the key schedule is invertible, we can recover all the subkeys used in ALE. Furthermore, we obtain $S_1 = E_K(0)$ and $K_1 = E_K(N)$, where K is the master key and N is the nonce. While we can not recover the master key, we have got enough information to encrypt and authenticate any message with nonce N.

Attack complexity. From Section 3 we have that the first local collision can be obtained in time from 2^{102} to 2^{119}, depending on the amount of available data (Section 3). However, for the second collision we are restricted to the same message. Hence we have to test possible differential trails one by one till we find one that yields the local collision. The complexity of this step is equal to that of the forgery attack with a single message — 2^{119}. As soon as both local collisions are constructed, the state recovery takes negligible time, as we only have to test 2^{24} state values conforming to the active S-boxes. The memory complexity is also negligible. The total time complexity equals 2^{120} forgery attempts of 48-byte messages.

5 Strengthening ALE

It is a natural question if ALE can be strengthened to prevent our attack. One may think that using five AES rounds would be enough, with the last round not extracting any values. Indeed, our trail would expand to a fully active state in

the final round. However, there is a 5-round trail with only 26 active S-boxes, of which 8 ones are extracted:

$$1 \xrightarrow{SR,MC} 4 \xrightarrow{SR,MC} 16 \xrightarrow{SR,MC} 4 \xrightarrow{SR,MC} 1 \xrightarrow{SR,MC} 4.$$

The total probability of the trail hence decreases to 2^{-110} (see Figure 4 for illustration). However, much fewer trails can be built for a single message. For each particular truncated differential trail we estimate with the rebound technique that for each set of extracted values there are 2^{14} valid trails. Hence the data complexity would be about 2^{96}. By playing with the trail layout and by adding one more active S-box we can further reduce it to about 2^{80}. Even though it violates the data restriction, the security margin seems to be quite thin. Adding one more round seem to solve the problem completely, as the best trail seems to have 22 active non-extracted S-boxes. Hence we believe that at least 6 rounds are required to counteract our attack.

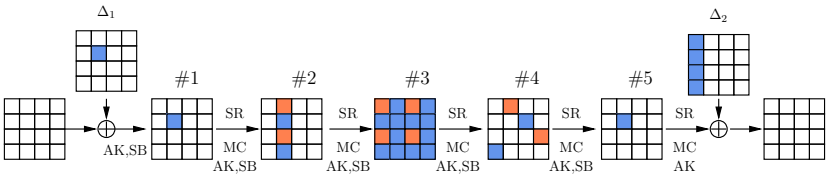

Fig. 4. Local collision trail for a 5-round variant of ALE.

Another countermeasure could be to decrease the number of extracted bytes. If only 3 bytes are extracted at each round, so that 12 bytes are injected, it might be difficult to construct a trail that yields a local collision. A much more elaborate analysis is needed to investigate this option. Still, it would give quite a penalty on the performance, but not that big as using 6 rounds instead of 4.

A third countermeasure could be to introduce key information into the round transformations which the aim to separate the leaked bytes from the S-boxes before and after the leak, as this has been done in ASC-1 [10]. This would affect the performance only very moderately, however depending on how exactly this key information would be derived, guess-and-determine extensions of the LOCAL approach would need to be considered as well.

6 Conclusion

We have demonstrated how to construct forgeries for ALE within the security claim limits. We show that the mere weight of a differential trail is a poor measure of the scheme resistance to differential attack as long as the values of active S-boxes are partially extracted or leaked. By choosing the trail values according to the extracted bytes, we can amplify its probability and eventually construct a

forgery using 2^{45} encrypted messages and 2^{110} time. The inability of the receiver in a general case to avoid the nonce reuse enables us to reconstruct the internal state of the encryption out of two forgeries on the same message, which in turn leads to the universal forgery attack. One can hence say that ALE, similarly to GCM, has high reforgeability [5].

We have also proposed several ways to strengthen ALE against our attack, which include a larger number of rounds and a different leakage scheme.

Model	**Data**	**Verification attempts**	**Memory**	**Security claim**
Forgery				
Known plaintexts	2^{102}	2^{102}	negl.	not violated
	2^{40}	2^{110}	negl.	violated
	1	2^{119}	negl.	violated
	1	1	negl.	violated, success rate 2^{-119}
State recovery, universal forgery				
Known plaintexts	1	2^{120}	negl.	violated

Table 1. Summary of attacks on ALE

References

1. `http://competitions.cr.yp.to/caesar.html`.
2. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT'00*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.
3. Alex Biryukov. The design of a stream cipher LEX. In *Selected Areas in Cryptography'06*, volume 4356 of *Lecture Notes in Computer Science*, pages 67–75. Springer, 2006.
4. Alex Biryukov and Dmitry Khovratovich. Related-Key Cryptanalysis of the Full AES-192 and AES-256. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009.
5. John Black and Martin Cochran. MAC reforgeability. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 345–362. Springer, 2009.
6. Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen, and Elmar Tischhauser. ALE: AES-based lightweight authenticated encryption. In *FSE'13, to appear*, 2013.
7. Florent Chabaud and Antoine Joux. Differential collisions in SHA-0. In *CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.

8. Joan Daemen and Vincent Rijmen. The Pelican MAC Function. *IACR Cryptology ePrint Archive*, 2005:88, 2005.
9. Orr Dunkelman and Nathan Keller. A new attack on the LEX stream cipher. In *ASIACRYPT'08*, volume 5350 of *Lecture Notes in Computer Science*, pages 539–556. Springer, 2008.
10. Goce Jakimoski and Samant Khajuria. ASC-1: An authenticated encryption stream cipher. In *Selected Areas in Cryptography'11*, volume 7118 of *Lecture Notes in Computer Science*, pages 356–372. Springer, 2011.
11. Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011.
12. Phillip Rogaway. Authenticated-encryption with associated-data. In *ACM Conference on Computer and Communications Security'02*, pages 98–107, 2002.
13. ISO/IEC 19772 JTC 1 SC 27. *Information technology – Security techniques – Authenticated encryption*, 2009.