ALE: AES-Based Lightweight Authenticated Encryption

Andrey Bogdanov[1], Florian Mendel[2], Francesco Regazzoni[3,4], Vincent Rijmen[5], and Elmar Tischhauser[5]

[1] Technical University of Denmark
[2] IAIK, Graz University of Technology, Austria
[3] ALaRI - USI, Switzerland
[4] Delft University of Technology, Netherlands
[5] Dept. ESAT/COSIC, KU Leuven and iMinds, Belgium

Abstract. In this paper, we propose a new *A*uthenticated *L*ightweight *E*ncryption algorithm coined *ALE*. The basic operation of ALE is the AES round transformation and the AES-128 key schedule. ALE is an online single-pass authenticated encryption algorithm that supports optional associated data. Its security relies on using nonces.
We provide an optimized low-area implementation of ALE in ASIC hardware and demonstrate that its area is about 2.5 kGE which is almost two times smaller than that of the lightweight implementations for AES-OCB and ASC-1 using the same lightweight AES engine. At the same time, it is at least 2.5 times more performant than the alternatives in their smallest implementations by requiring only about 4 AES rounds to both encrypt and authenticate a 128-bit data block for longer messages. When using the AES-NI instructions, ALE outperforms AES-GCM, AES-CCM and ASC-1 by a considerable margin, providing a throughput of 1.19 cpb close that of AES-OCB, which is a patented scheme. Its area- and time-efficiency in hardware as well as high performance in high-speed parallel software make ALE a promising all-around AEAD primitive.
Keywords: authenticated encryption, lightweight cryptography, AES

1 Introduction

Motivation. As essential security applications go ubiquitous, the demand for cryptographic protection in low-cost embedded systems (such as RFID and sensor networks) is drastically growing. This necessitates secure yet efficiently implementable cryptographic schemes. In such use cases, the area and power consumptions of a primitive in hardware are usually of paramount importance and standard solutions are often prohibitively costly to deploy.

Once this problem was recognized, the cryptographic community was fast to address it by proposing a great deal of specialized lightweight cryptographic algorithms, which include stream ciphers like Trivium [17], Grain [25], and Mickey [5], block ciphers like SEA [38], DESL, DESXL [30], HIGHT [26], mCrypton [31], KATAN/KTANTAN [16], and PRESENT [10], and hash functions like Quark [4],

Photon [24], and Spongent [9] — to mention only some fraction of them. We note that the latter hash functions are following the overall design strategy of a permutation-based sponge construction [6], similarly to Keccak [7], which also provides competitive lightweight properties [29].

However, when it comes to *authenticated encryption* — the fundamental security functionality in most real-world security systems — one has to establish that, rather surprisingly, only a few lightweight schemes have been proposed so far, examples are Grain-128a [2] and Hummingbird-2 [21]. At the same time, message secrecy – as provided by plain encryption – is often of limited value in practice if not accompanied by message authentication. This stipulates the acute need for authenticated encryption in the field which is reflected in NIST [20] and ISO/IEC [1] documents on modes of operation for block ciphers.

In the context of lightweight cryptography though, these standard modes of operation have significant practical limitations. First, the lightweight block ciphers are usually designed to save on state bits, so that the block size and key size are usually kept at the edge of the reasonable minimum (it is rather typical in lightweight cryptography to propose a block cipher with a 64-bit block and a 80-bit key). This significantly confines the security level of modes of operation theoretically attainable due to generic attacks. Second, the standard authenticated-encryption modes of operation traditionally aim at high-speed implementations by minimizing the number of block cipher calls and other operations one has to perform per data block processed. For example, OCB [36], which clearly outperforms such wide-spread schemes as AES-GCM and AES-CCM in standard software, requires essentially a single AES call per data block at bulk encryption only. However, such modes usually do not pay too much attention to the amount of memory and the circuit size one needs in a lightweight hardware implementation. For instance, AES-OCB requires at least four 128-bit registers and both AES-encryption and -decryption engines for both encryption/authentication and decryption/verification. Besides, OCB is a patented scheme which hampers its wide deployment in the field.

A straightforward solution would be to address the first limitation (small state) by raising the total internal state size of the lightweight primitives, for instance, to 256 bits to avoid generic attacks up to a bound of 2^{128} operations. However, this would in turn take away their major source of advantage and make their area occupation comparable to that of AES-128. That is why we feel that a dedicated authenticated-encryption design can also be based on AES when 128-bit level of security is desired.

In an attempt to mitigate the second limitation (additional memory requirements imposed by modes) one might choose to go for encrypt-then-mac or mac-then-encrypt. However, not only would it jeopardize the highly relevant implementation goal for the scheme to be online but also require double message input (being essentially two-pass) and twice more operations per data block than e.g. OCB. In general, there appear to be no single-pass authentication encryption modes of operation for block ciphers preserving the minimal state size

required. This emphasizes the demand for a dedicated lightweight authenticated-encryption design.

Moreover, we also want to make this new design fast in software, as opposed to some bit-oriented lightweight ciphers (such as Grain, Trivium, KATAN, PRESENT, etc.) which succeed in attaining a low area in hardware but whose performance in software is not even comparable to that of AES, especially in the presence of the Intel AES-NI instructions. We feel that not much efficiency can be gained by designing a slightly more efficient *generic* authenticated-encryption mode of operation for block ciphers since the bottleneck will remain the one block cipher call per data block. This is not only true for authenticated encryption modes but also for MAC-only and encryption-only modes.

The situation is, however, essentially different if the designer is allowed to look inside the specific underlying block cipher such as AES and to construct a dedicated mode of operation which only uses exactly as many operations of the underlying block cipher as needed. This is the approach taken in the designs of the stream cipher LEX [8] and the message authentication algorithm Pelican [14]. Lately, similar reasoning was applied to the setting of authenticated encryption resulting in the design of ASC-1 [28].

ALE. This paper proposes a lightweight authenticated encryption algorithm based on AES called *ALE* (*A*uthenticated *L*ightweight *E*ncryption) which is efficient both in hardware and software. It is a single-pass nonce-based online scheme that preserves the memory alignment of data. The design of ALE combines some ideas of Pelican, LEX and ASC-1 in a lightweight manner. In a nutshell, the algorithm uses Pelican keyed in all rounds (similarly to PC-MAC [34]) for computing the authentication tag and leaks bytes of the state in every round in a LEX-type way for encryption/decryption. It has a 256-bit secret internal state dependent on both key and nonce.

By requiring only 2.5 kGE of area in lightweight ASIC hardware, which is less than 100 GE overhead compared to plain AES-ECB in the smallest implementation available [35], ALE is about half the size of AES-OCB and ASC-1. In terms of speed in the lightweight implementation for medium-size messages and longer, ALE is about 2.5 times faster than AES-OCB and about 4.5 times faster than ASC-1 in its smallest implementation. When using the parallel AES-NI instructions, ALE outperforms AES-GCM, AES-CCM and ASC-1 by a considerable margin, providing a throughput close to that of AES-OCB, which is a patented scheme.

At a first glance, the overall design philosophy of ALE might seem similar to that of ASC-1. However, as the numbers of relative area and speed above already strikingly suggest, ASC-1 has several crucial shortcomings in the way of practical implementation. First, ASC-1 needs an internal state which is twice larger than that of ALE, which accounts to the significant difference in area requirements. Second, the non-sequential order in which the AES-256 subkeys are used in ASC-1 (e.g. subkey 11 is needed already in the first round) combined with its serial nature, has a considerable impact on its performance. In lightweight

hardware, the engine in adjacent operations has to have subkeys which are many rounds apart which can be done either by computing a key state back and forth (which costs time) or by storing some values high-speed software (which costs area). In high-performance parallel software implementations, the subkeys have to be computed on the fly (since the key state is evolving) beforehand and stored in registersto aviod additional memory accesses, which contrains the advantage of using AES-NI instructions a lot. Finally, ASC-1 does not accept associated data that can be vital in some networking settings while ALE explicitly deals with it.

The remainder of the paper is organized as follows. Section 2 gives a specification of the algorithm. Section 3 introduces some elements of its cryptanalysis. Section 4 provides lightweight implementation numbers of the algorithm in ASIC hardware. Section 5 implements ALE in software using AES-NI instructions on a SandyBridge Intel processor. We conclude in Section 6.

2 The Authenticated Lightweight Encryption (ALE) algorithm

In this section, we describe ALE – our new authenticated lightweight encryption algorithm. The basic operation of ALE is the AES round transformation and the AES-128 key schedule. In all the following, we assume that the reader is familiar with AES.

2.1 Specification

ALE is an online single-pass nonce-based authenticated encryption algorithm with associated data. Its encryption/authentication procedure accepts a 128-bit master key κ, a message μ, associated data α and a 128-bit nonce $\nu \neq 0$. An equivalent of at most 2^{48} bits are allowed to be authenticated or both authenticated and encrypted with the same master key. The encryption/authentication procedure outputs the ciphertext γ of exactly the same bit length as the message μ and the authentication tag τ of 128 bits for both the message μ and associated data α. Its decryption/verification procedure accepts key κ, ciphertext γ, associated data α, nonce ν and tag τ. It returns the decrypted message μ if tag is correct or $\perp$ otherwise.

The encryption/authentication operation can be described in five steps:

Padding: The padding of ALE is similar to the one of the MD4 hash function. First a "1" is appended to the message μ, followed by ℓ "0" bits (with $\ell = 128 - (|\mu| + 1 + 64 \pmod{128})$, and finally the message length $|\mu|$ coded on 64 bits is appended. The resulting padded message M is split into t blocks of 128 bits each, $M = m_1 \| \ldots \| m_t$. Note that for associated data the same padding method is used and the padded associated data is split into r blocks of 128 bit each, $A = a_1 || \ldots || a_r$.

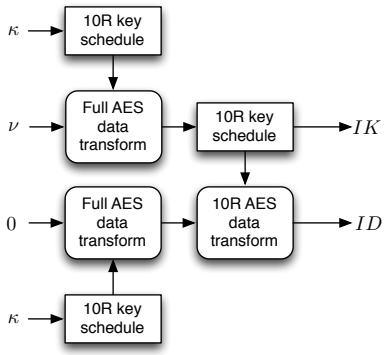

Fig. 1. Initialization of ALE.

Initialization: The internal state consists of two 128-bit states: the key state (upper line in Figure 1) and the data state (lower line in Figure 1). The key state is initialized with nonce ν encrypted with AES[6] under the master key κ. The data state is initialized in two steps, First, it is initialized with 0 encrypted with AES under the user-supplied key κ. Second, the result $\mathrm{AES}_\kappa(0)$ is AES-encrypted using the initialized key state as key. After that, the final subkey of the last AES encryption is updated one more time using the AES round key schedule with byte round constant x^{10} in $\mathbb{F}_{2^8}$. This value is stored in the key state. Now both states are initialized.

Processing associated data: If there is only one padded associated data block, then a_1 is xored to the data state and one proceeds with processing message immediately. Otherwise, if there are at least two padded associated data blocks, A is processed block by block: The data state is encrypted with 4 rounds of AES using the key state as key. The final round subkey is updated one more time using the AES round key schedule with byte round constant x^4 in $\mathbb{F}_{2^8}$. This value is stored in the key state. The next block of A is xored to the data state.

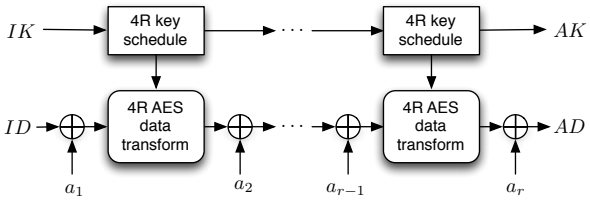

Fig. 2. Processing associated data in ALE.

[6] Here and further in the paper, we imply AES-128 whenever we write AES.

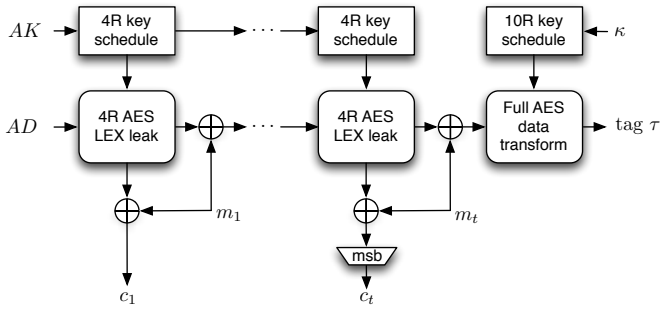

Fig. 3. Processing message and finalization in ALE.

Processing message: M is processed block by block: The data state is encrypted with 4 rounds of AES using the key state as key. 16 bytes are leaked from the data state in the 4 rounds of AES in accordance with the LEX specification.

This leak is xored to the current block of M. The final round subkey is updated one more time using the AES round key schedule with byte round constant x^4 in $\mathbb{F}_{2^8}$. This value is stored in the key state. The current block of M is xored to the data state.

For the last block of M the exact required number of most significant bits are taken from the leak and xored to the last block (without padding) to produce the last bits of ciphertext, and m_t is xored to the data state.

Finalization: The data state is encrypted with the full AES using the master key κ. The output of this encryption is returned as the authentication tag τ for the message and associated data.

The decryption/verification procedure is defined correspondingly. The only two differences are that one works with the ciphertext $\gamma = c_1||\ldots||c_t$ instead of the message μ while xoring with the stream and that the supplied tag value τ is compared to the one computed by the algorithm. We want to stress that only if the tag is correct the decrypted message is returned.

2.2 Security assumptions and claims

The security analysis of the algorithm starts from the following two assumptions.

Assumption 1 (Nonce-respecting adversary) *A nonce value is only used once with the same master key for encryption.*

This assumption is quite common among nonce-based designs. Note that on most platforms, this assumption can be easily satisfied by implementing the nonce as a counter.

Assumption 2 (Abort on verification failure) *If the verification step of the algorithm reveals that the ciphertext has been tampered with, then the algorithm returns no information beyond the verification failure. In particular, no plaintext blocks are returned.*

This assumption significantly reduces the impact of chosen-ciphertext attacks, since the adversary obtains very little information from a chosen-ciphertext query. We feel that this assumption is quite natural for authenticated encryption modes. After all, when the verification fails, we know that the integrity of the plaintext has been jeopardised, and there is no reason to output it. The assumption does, however, exclude implementations where decryption is done in a streaming mode, since all plaintext blocks need to be kept inside until the verification has completed successfully.

Under these assumptions, the security claims for the algorithm are as follows.

Claim 1 (Resistance against state recovery) *Any internal state recovery with complexity equivalent to processing N data blocks has a success probability at most $N2^{-128}$.*

Claim 2 (Resistance against key recovery) *Any key recovery with complexity equivalent to processing N data blocks has a success probability at most $N2^{-128}$, even if the internal state has been recovered.*

Claim 3 (Resistance against forgery w/o state recovery) *Any forgery attack not involving key recovery/internal state recovery has a success probability at most 2^{-128}.*

2.3 Properties

Here we list some of ALE's merits in terms of implementation. Since ALE is based on similar design principles as Pelican MAC and LEX, it also shares many strong properties of these two designs.

- Security analysis benefits from existing analysis on AES as well as Pelican MAC and LEX.
- AES hardware/software implementations might be reused with only a few simple modifications, including the usage of Intel AES instructions.
- Side-channel attack countermeasures developed for the AES will be useful for ALE as well, including threshold implementations in hardware to thwart first-order power- and EM-based differential attacks and bitsliced implementations to mitigate cache-timing leakage.
- For long messages, ALE needs only about 4 AES rounds to both encrypt and authenticate a block of message, which is similar to ASC-1. However, about 10 AES rounds are needed by AES-OCB and 20 AES rounds are required by AES-CCM to process a data block.

– The overhead of ALE per message amounts to 3 AES calls. This is less than the overhead of ASC-1 which is 4 AES calls but more than the overhead of AES-OCB of 2 AES calls. AES-CCM has virtually no overhead but has an excessive cost per block.
  However, in terms of the number of AES rounds, AES-OCB becomes less efficient already for messages longer than 128 bits and ASC-1 is always inferior to ALE. Note that AES-OCB contains a nonce stretching mechanism that effectively saves one AES call overhead if multiple messages are encrypted with the same key and with adjacent counter values as nonces. However, implementing this mechanism in lightweight hardware requires an additional 128-bit state which increases the area requirement by another 700-800 GE.
– Only two 128-bit states are needed by ALE to implement encryption, which makes it lightweight-friendly. At the same time, 4 states needed for AES-OCB and ASC-1. AES-CCM requires 3 states. In fact, 2 states needed by ALE are even less than the encryption-only AES-CTR occupies, where some space needs to be allocated for the counter.
– Only the AES encryption engine is needed by ALE for both the encryption/authentication and decryption/verification procedures. AES-OCB requires both encryption and decryption engines for supporting those operations.
– ALE is an online scheme meaning it is single-pass and does not have to know message length before the last message block is input. AES-CCM is off-line. Additionally AES-CCM is two-pass. AES-OCB and ASC-1 are also online schema.
– ALE accepts associated data while ASC-1 does not. AES-CCM and AES-OCB can both work with associated data. AES-OCB is additionally capable of accepting static associated data (which does not require any recomputation for a new nonce).

3 Security analysis

Since ALE combines some ideas of Pelican MAC [14] and LEX [8] it benefits from existing security analysis. In the following, we briefly recall existing security analysis on these two primitives and discuss their relevance to ALE with respect to its Claims 1,2 and 3.

3.1 Forgery without State Recovery

Like any MAC derived from the ALRED construction [15] also Pelican MAC enjoys some level of provable security. It is shown that, in the absence of internal collisions, the security of the construction can be reduced to the security of the n-bit underlying block cipher [15, Theorem 1, Theorem 2]. In other words, Pelican cannot be broken with less than $2^{n/2}$ queries unless the adversary also breaks the block cipher itself. However, the security proofs of Pelican MAC rely on the fact that the iteration function is unkeyed. Therefore, they don't carry over

to ALE. PC-MAC is another MAC function derived from the original design [34]. PC-MAC uses a keyed iteration function and has a proof of security in the indistinguishibility framework.

For the Pelican MAC two approaches to exploit knowledge of the (unkeyed) iteration function are described by the designers to generate internal collisions and hence forgeries.

Fixed points. Since the iteration function is known in Pelican MAC, one may compute the number of state values that are resulting in fixed points for a given message block m_i. Assume the number of fixed points is x, then the probability that inserting the message block m_i in a message will not impact its tag and hence result in a forgery is $x \cdot 2^{-n}$.

However, if the iteration function can be modeled as a random permutation, then the number of fixed points has a Poisson distribution and is expected to be small [15]. Moreover, in ALE the iteration function is keyed with a nonce dependent session key. Since this key changes for every iteration function, one needs a fixed point in both the key state and data state rendering the attack inefficient.

Extinguishing differentials. Extinguishing differentials are very similar to differential cryptanalysis for block ciphers. The main idea is to find pairs of messages (or in our case also ciphertexts) with a certain difference that may result in a zero difference in the state with a high probability after the difference has been injected.

However, in the case of Pelican MAC the iteration function consists of 4 rounds of Rijndael implementing the wide trail design strategy [13], which allows to prove good bounds against differential attacks. In more detail, any differential characteristic spanning over 4 rounds has at least 25 active S-boxes resulting in an upper bound for the differential probability of 2^{-150}. Moreover, the differential probability of any differential can be upper bounded by 2^{-114} [27]. Note that this is not far away from the theoretically optimal bound of $2 \cdot 2^{-128}$. In other words, Pelican MAC and hence also ALE provides good upper bounds for the probability of extinguishing differentials.

Moreover, we want to note that in ALE the iteration function is keyed with a nonce dependent session key, which is changed for every encryption/authentication procedure, complicating the application of differential cryptanalysis to ALE, since an attacker also needs to predict the differences in the session key. However, since this session key is generated by encrypting the nonce with the master key using 10 rounds of AES, this seems to be a very difficult task.

3.2 State Recovery

All published attacks [11,19,40] on Pelican MAC so far are forgery attacks making use of (generic) internal collisions on the internal state. Note that due to the small state size of Pelican MAC of 128 bits, internal collisions can be found

with complexity of 2^{64} due to the birthday paradox. However, in ALE a nonce dependent session key is used making it difficult to detect internal collisions on the state unless the session key collides as well, basically doubling the internal state size and giving some reinsurance in the design.

Most attacks on LEX published so far use the fact that LEX uses the same round keys repeatedly. For instance the main idea of the key-recovery attack in [11,18] is to find a pair of internal states after different numbers of encryptions that partially collides after 4 rounds. Since the round keys are reused in LEX, the adversary can easily locate two states that collide in the part of the state which contains the round key. Hence, the complexity of a brute-force search for (partial) internal collisions is determined by the size of the part of the state that contains the ciphertext, i.e. 128 bits.

In ALE however, the round keys are not reused. Hence, the complexity of a brute-force search for (partial) internal collisions is determined by the size of the full internal state, i.e. 256 bits. It follows that finding (partial) internal collisions becomes more difficult rendering the attack infeasible. Note that even if the attack would be applicable, it might only be used to recover the internal state, but not the master key.

If Assumption 1 is not satisfied, i.e. if nonce values are used repeatedly, then the round keys are repeated. In that case, the attacks on Pelican and LEX can be extended and applied to ALE. Because ALE combines injection and extraction, the attacks become more powerful, and security is lost.

3.3 Key-Recovery

To recover the master key κ in ALE an attacker needs to break the initialization of ALE. However, even though assuming that the full internal sate after the initialization is known to the attacker, he still needs to break full (10 rounds) of AES to recover the master key.

3.4 Additional Security Analysis

Distinguishing attacks. The encryption component of ALE is inspired by the stream cipher LEX. The keystream bits in LEX are generated by extracting 32 bits from each round of AES in the OFB mode. In [22], Englund et al. describe a distinguishing attack that is applicable to block ciphers in the OFB mode in general. To be more precise, whenever the part of the state that depends on both the key and the IV is smaller than twice the key size (as it is the case for instance in LEX) the attack theoretically succeeds. However, in LEX the attack is thwart by limiting the number of keystream bits that can generated from one master key. In ALE we have a similar restriction but more important the internal state is larger due to the session key (depending on the nonce and the master key) used to key the iteration function.

Slide attack. A slide attack for an earlier version of LEX has been found by Wu and Preneel in [39] and fixed later by Biryukov in a new version of LEX. To avoid slide attacks two different functions for the initialization and the encryption/authentication should be used. Even a very small difference between the two is sufficient. For instance, the new version of LEX uses the full AES with the XOR of the last subkey for the initialization and AES without the XOR of this subkey for the encryption.

However, ALE uses the full AES (10 rounds without the application of MixColumns in the last round) in the initialization, but only 4 rounds of AES for the encryption/authentication. We think that this is sufficient to break the similarities used by slide attacks.

4 Lightweight ASIC hardware implementation

This section presents a lightweight ASIC hardware implementation of ALE, and compares it to the existing authentication encryption schemes such as AES-OCB, AES-CCM and ASC-1. In this section, we did not include the results obtained with AES-GCM as this schema is not particularly suitable for low cost hardware. In fact, it requires an extra module for implementing the Galois field multiplication which, additionally, has to be invoked several times.

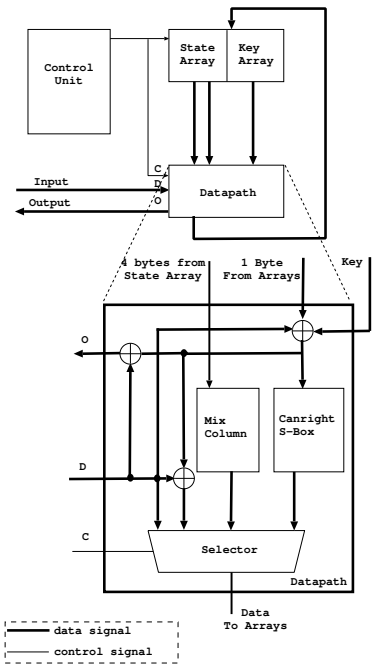

Fig. 4. The lightweight implementation of ALE

4.1 Hardware architecture

ALE was implemented targeting the lowest ASIC area occupation possible. For this reason, our hardware architecture is based on the most compact AES implementation published so far [35]. The original AES implementation has a mixed data-path: it instantiate a single S-box following the proposal of Canright [12] but performs the MixColumn on all the 32 bit in parallel.

The overall design is depicted in Figure 4. The base AES design was extended to support our authenticated encryption proposal. A more complex control unit was developed to handle the padding, the initialization and finalization, the LEX-type leak, and the xor of the state with the input message. Also, a number of multiplexers where added to the architecture to correctly select the inputs of the AES accelerator. Our implementation requires to load two times the key and one time the nonce and the null vector. The nonce is loaded in the first execution of AES, while the null vector is loaded in the second. The key is loaded once during the first execution of the AES and once during the last execution of AES. Also, the input and output values should be maintained in the respective wires and synchronized with the operations of the accelerator on order to correctly perform the additions with the message and the additional data.

4.2 Comparison

We implemented all the designs having the same design goals and using the same lightweight AES engine in their core. All the considered schemes were described in VHDL and then synthesized using the tool Synopsys design compiler 2009.06. We performed a number of synthesis targeting different technologies (90nm, 65nm, and 45nm), frequencies, and optimization parameters of the synthesis tool.

Table 1 summarizes the implementation numbers (including area and timing) of ALE as compared to the reference designs. These results were obtained setting the clock frequency to 20 MHz and using the STMicroelectronics 65nm CMOS technology and the corresponding standard cell library characterized for LP-HVT (low power high Vt) process. This technology was the one which exhibits the best trade off between area and power consumption, thus the one which resulted more suitable for lightweight applications. The clock frequency was set to 20 MHz as usually, in low-cost hardware applications, the speed constrain is very relazed (contrary to the area and power consumption). However, during the whole set of experiments, we successfully synthesized our designs with a clock frequency of up to 200 MHz.

As we were targetting low-cost hardware, we also report clocks per byte and provide a graph for different message lengths in Table 2 and Figure 5. The cycle count does not consider the overhead for loading and offloading of the data. As indication, Table 1 reports also the power consumption of each algorithm. The estimation was carried out with Synopsys power compiler 2009.06, using the standard tool parameters for the switching activity.

Table 1. Lightweight ASIC implementation numbers for ALE compared to AES-OCB2, AES-CCM and ASC-1. Overhead indicates the number of cycles needed for the initial setup and the finalization of the authenticated encryption. The net per block provides the number of clock cycles required to process each block of data, on top of the overhead per message. The designs marked with 'e/d' incorporate both encryption/authentication and decryption/verification functionalities.

Design	Area (GE)	Net per 128-bit block (clock cycles)	Overhead per message (clock cycles)	Power (uW)
AES-ECB	2,435	226	-	87.84
AES-OCB2	4,612	226	452	171.23
AES-OCB2 e/d	5,916	226	452	211.01
ASC-1 A	4,793	370	904	169.11
ASC-1 A e/d	4,964	370	904	193.71
ASC-1 B	5,517	235	904	199.02
ASC-1 B e/d	5,632	235	904	207.13
AES-CCM	3,472	452	-	128.31
AES-CCM e/d	3,765	452	-	162.15
ALE	**2,579**	**105**	678	94.87
ALE e/d	**2,700**	**105**	678	102.32

ALE occupies 2,581 GE, and requires 783 clock cycles to authenticate and encrypt one block of 128 bits. This cycle count includes the overhead of 678 cycles, which is caused by the three invocation of the AES algorithm needed for initialization and finalization. Thus, only 105 clock cycles are needed to process each further 128-bit block of data.

As comparison, we report in Table 1 also the performances of the AES core used as starting point (AES-ECB) of our implementation and the ones of AES-OCB, ASC-1, and AES-CCM authentication encryption schema. All the algorithms were implemented using the same lightweight AES engine [35] and the same experimental setup as ALE.

In its most compact implementation, ASC-1 is especially slow due to its complex non-serial key schedule which has a high overhead (for instance, one has to know the 11th key of the AES-256 expanded key to compute the first round and the first key again to compute the 5th round) which makes backward and forward computations necessary. This implementation is given in Table 1 as ASC-1 A. However, The key schedule overhead can be reduced if an additional 128-bit register is introduced. This implementation is given referred to as ASC-1 B in the table.

It can be observed that the overhead for the support of both the encryption/authentication and decryption/verification functionalities (those implementations are marked as 'e/d' in Table 1) is fairly small for ALE and ASC-1. At the same time, since AES-OCB additionally requires an AES decryption engine for decryption, the overhead is much more significant there.

ALE occupies an area which is approximately two times lower than those of AES-OCB and ASC-1, while providing an overall speed at least two times higher, being particularly suitable for lightweight applications. ALE also nicely compares with the results reported in literature for Hummingbird-2 [21], which in its smallest version has an area of approximately 2,159 (estimated using a different technological library), and Grain-128a with authentication [2] which occupies approximately 2,770 gates (estimated by the designers).

Table 2. Lightweight ASIC implementation numbers for ALE compared to AES-OCB2, AES-CCM and ASC-1 (in clocks per byte)

	message length (bytes)									
Algorithm	16	32	64	128	256	512	1024	2048	4096	8192
ECB	14.12	14.12	14.12	14.12	14.12	14.12	14.12	14.12	14.12	14.12
OCB2	42.38	28.25	21.19	17.66	15.89	15.01	14.57	14.35	14.24	14.18
ASC-1 A	79.62	51.38	37.25	30.19	26.66	24.89	24.01	23.57	23.35	23.24
ASC-1 B	71.19	42.94	28.81	21.75	18.22	16.45	15.57	15.13	14.91	14.80
CCM	28.25	28.25	28.25	28.25	28.25	28.25	28.25	28.25	28.25	28.25
ALE	48.93	27.75	17.15	11.85	9.21	7.88	6.20	6.89	6.72	6.64

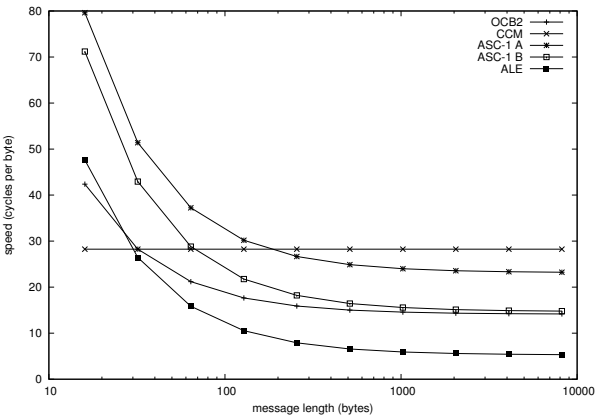

Fig. 5. Hardware performance of ALE with respect to other AES-based authenticated encryption schemes for different message lengths in lightweight ASIC implementations

5 High-Performance Software Implementation

In this section, we evaluate the software performance of ALE and compare it to other authenticated encryption schemes based on the AES. We propose such evaluation because, as pointed out by Matsuda and Moriai [32], lightweight algorithms will be used in the sensors which will populate the internet of things. After the collection, the data will be forwarded to the servers of the cloud, where the same algorithm, this time implemented to achieve high performances, will be used for decryption.

5.1 The Setting

First, we need to establish a common setting for the performance evaluation of all algorithms in order to have a fair platform for comparison. We base our scenario on the following assumptions.

Message lengths. In most communication protocols, typical messages are relatively short, rarely exceeding 1024 bytes [33]. It is therefore of great importance to include initialisation overhead and take measurements for messages consisting of only one or a few blocks. At the same time, performance usually starts to saturate with messages of two or four kilobytes. We therefore provide data for $16 \cdot 2^b$ bytes, with $0 \leq b \leq 10$. Furthermore, we assume that these 2^b blocks encompass the already padded message.

Parallel processing of messages. The fact that most processed messages are rather short suggests that many of them will typically be encrypted under the same key, but with a different nonce. This implies that a high-performance software implementation can benefit from processing multiple messages for different nonces in parallel. Note that the processing of messages with several different keys and different nonces can be parallelised in the same way.

AES-NI and pipelining. Since we deal with AES-based ciphers, high performance software implementations means using the AES-NI instructions [23]. The most critical factor in achieving good performance with AES-NI is to fully utilise the pipeline, which is 8 cycles for the Sandy Bridge microarchitecture, for which our implementations are optimised.

5.2 Implementation

According to this common scenario, we have implemented the following authenticated encryption schemes: CCM, GCM and OCB3 with AES as the underlying block cipher; ASC-1, and ALE. As a base line, we also include the unauthenticated modes ECB and CTR. The used OCB3 implementation is the most recent reference implementation from [37], the GCM implementation is the one of OpenSSL v1.0.1c.

Those algorithms that are not inherently parallelisable (CCM and ALE) were implemented following the paradigm of processing multiple messages with different nonces in parallel, with CCM processing two independent messages in

parallel, and ALE four. ASC-1 was found not to benefit from this, since the overhead introduced by its key schedule already requires storing key material in cache memory due to the limited number of 128-bit registers. As an example, its key scheduling requires the use of the 11th key already in the first round, and the first key again in the 5th, and so forth. For OCB3, we include the overhead introduced by calculating the initial key- and nonce-dependent values. However, we do employ nonce-spreading to avoid the initial block cipher call most of the time.

For the implementation of ALE, we only have four AES rounds per message block instead of ten for the authenticated modes. However, this also means that in order to fill the 8 pipeline stages, we would have to calculate 8 key schedule updates. Using the native AESKEYGENASSIST instruction for this purpose actually decreases the performance, since it not pipelined in the same way as the AES round functions.

With four messages processed in parallel, we can however avoid using AESKEYGENASSIST by using AESENCLAST for the S-box step for the four AES keys, and doing the key schedule's LFSR manually, but in parallel on 128 bits. This implies that we can at most issue 5 AES round instructions every 8 cycles. See the pseudocode below:

```
loop:
  # ... combine 4*32 bits from key1, ..., key4 in keyblock
  aesenclast keyblock, 0
  aesenc     state1, key1
  aesenc     state2, key2
  aesenc     state3, key3
  aesenc     state4, key4
  # ... LEX leaks
  # ... inverse ShiftRows on keyblock
  # ... spread keyblock to key1, ..., key4
  # ... key schedule LFSR parallel on key1, ..., key4
  goto loop
```

We also note that in a serial (as opposed to high-performance) implementation, ALE has the distinct advantage over OCB3 by only requiring 4 AES round plus key scheduling per block, in contrast to full 10 AES rounds.

5.3 Results

All measurements were taken on a single core of an Intel Core i5-2400 CPU at 3100 MHz, and averaged over 100000 repetitions. Our findings are summarised in Table 3 and illustrated in Figure 6.

One can see that while the initialisation overhead generally has a huge impact on the performance, this effect starts to fade out already at messages of around 256-512 bytes. Due to the parallel processing used for CCM, it almost ties with GCM for medium-size and larger messages. OCB3 achieves nearly optimal performance starting from 512 byte message length due to its parallelisability which enables it to fully utilise the eight pipeline stages and compensates for its

Table 3. Software performance of authenticated encryption schemes based on the AES. The platform is Intel Sandy Bridge (AES-NI). All numbers are given in cycles per byte (cpb). A star (*) indicates that this implementation is processing multiple messages in parallel (for inherently serial algorithms for which this results in a performance increase).

	message length (bytes)						
Algorithm	128	256	512	1024	2048	4096	8192
ECB	1.53	1.16	0.93	0.81	0.75	0.72	0.71
CTR	1.61	1.22	0.99	0.87	0.80	0.77	0.76
CCM*	3.97	3.49	3.31	3.22	3.18	3.15	3.15
GCM	4.95	3.88	3.33	3.05	2.93	2.90	2.89
OCB3	2.69	1.79	1.34	1.12	1.00	0.88	0.86
ASC-1	7.74	4.80	3.69	2.88	2.78	2.64	2.61
ALE*	3.55	2.34	1.74	1.44	1.31	1.23	1.19

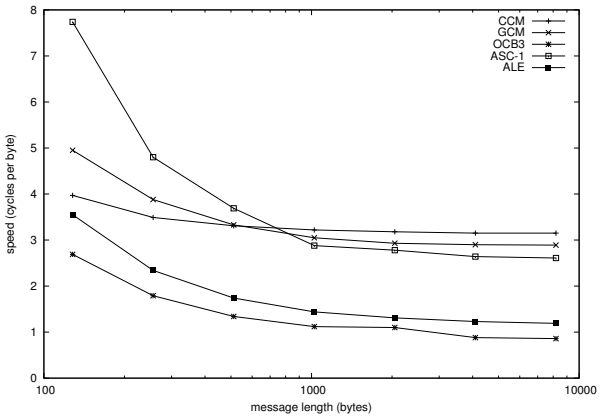

Fig. 6. Software performance of AES-based authenticated encryption schemes for different message lengths on Intel Sandy Bridge (AES-NI).

initialisation overhead. ASC-1 generally performs slower, mostly due to its non-sequential use of the AES-256 key schedule which requires additional storage of key material, exceeding the available 128-bit registers.

The experimental findings can be summarised as follows: When implemented for multiple message processing, ALE provides software performance quite close to OCB3-AES, and significantly better performance than CCM, GCM or ASC-1. In [33], McGrew estimates that for high-speed data links, authenticated encryption with a throughput of up to 100 GBit/s would be desirable. In view of the results of [3], AES-NI instructions benefit from a practically linear speed-up on multiple cores. At the moment, standard Sandy Bridge desktop processors are available with 6 cores at a frequency of 3.1 GHz. For messages of 1KB length, this implies a throughput of 103.3 GBit/s with ALE, and 132.8 GBit/s with OCB3. This means that the above-mentioned performance requirement can be fulfilled with either ALE or OCB3 using only one standard desktop CPU, with ALE having the advantage of not being patented.

6 Conclusion

In this paper, we have proposed ALE – a new Authenticated Lightweight Encryption algorithm based on AES. It is a single-pass nonce-based online scheme that combines some ideas of Pelican MAC, LEX and ASC-1 in a highly lightweight manner. ALE is about half the size of ASC-1 and in terms of speed in the lightweight implementation, it is about 4.5 times faster than ASC-1 in its smallest implementation.

By requiring only 2.5 kGE of area in lightweight ASIC hardware ALE is actually significantly smaller than most other authentication encryption modes including the popular modes AES-OCB and AES-CCM. In terms of speed in the lightweight implementation, ALE is about 2.5 times faster than AES-OCB and about 5 times faster than AES-CCM. When using the parallel AES-NI instructions, ALE outperforms AES-GCM, AES-CCM and ASC-1 by a considerable margin, providing a throughput close to that of AES-OCB, which is a patented scheme.

Acknowledgments. The authors thank Axel Poschmann for providing the reference implementation of the AES algorithm. Part of this work was done while Andrey Bogdanov and Florian Mendel were with KU Leuven. The work has been supported in part by the Austrian Science Fund (FWF), project TRP 251-N23 and by the Research Fund KU Leuven, OT/08/027.