Efficient Design Strategies Based on the AES Round Function

Jérémy Jean(✉) and Ivica Nikolić

Nanyang Technological University, Singapore, Singapore
Jean@pmail.ntu.edu.sg

Abstract. We show several constructions based on the AES round function that can be used as building blocks for MACs and authenticated encryption schemes. They are found by a search of the space of all secure constructions based on an efficient design strategy that has been shown to be one of the most optimal among all the considered. We implement the constructions on the latest Intel's processors. Our benchmarks show that on Intel Skylake the smallest construction runs at 0.188 c/B, while the fastest at only 0.125 c/B, i.e. five times faster than AES-128.

Keywords: Fast software implementation · AES · AES-NI · Skylake

1 Introduction

As a block cipher standard, the AES has inspired many cryptographic designs. Stream and block ciphers, authenticated encryption schemes (AEs), cryptographic hash functions and Message Authentication Codes (MACs) based on the AES benefit from its two main features, namely, its security and efficiency. The security benefit is twofold. First, as the AES is the most popular block cipher, it has been extensively analyzed and its security is well understood [9,14,15]. Second, the AES is based on the so-called wide-trail strategy [6], which provides resistance against the standard differential and linear attacks. The efficiency benefit is significant as well. Due to its internal structure, the AES allows fast software implementations based on look-up tables as well as even more efficient bit-sliced implementations [12]. Furthermore, the latest mainstream processors have a dedicated set of instructions, called AES-NI, that provides a complete implementation of the AES. These handy instructions allow with a few lines of code to execute one block cipher call with exceptionally high efficiency (measured in cycles per byte of data or c/B). For instance, on the same architecture, the table-based implementation of AES-CTR runs at around 10 c/B, its bit-sliced implementation at around 7.5 c/B, while its AES-NI implementation at less than 1 c/B. As significant speedups are observed when AES-NI are available, it is important to understand how far we can benefit from them.

Depending on the security requirements and adversarial model, designs based on the AES may use round-reduced version of the block cipher. For instance, Pelican-MAC [8], Alpha-MAC [7], LEX [1], ASC-1 [11], and ALE [3], use only

T. Peyrin (Ed.): FSE 2016, LNCS 9783, pp. 334–353, 2016.
DOI: 10.1007/978-3-662-52993-5_17

four rounds of the AES to process one message block (cf. to the ten rounds in the original AES-128 block cipher). Obviously, the reduction in the number of rounds has a direct impact on the efficiency and these designs run at much higher speed. The decision to reduce the number of rounds to four stems from the wide-trail strategy, since in some cases four rounds already provide sufficient level of security. Only a few designs use less than four rounds, as the security analysis becomes more intricate.

Our Contributions. We examine AES-based constructions that can be used as building blocks of secret-key primitives (e.g., MACs and authenticated encryption schemes). Our main goal is to push the limits of efficiency of constructions that can be implemented with the AES-NI, without sacrificing their security.

As reference points and benchmarks, we use the two authenticated encryption schemes AEGIS-128L and Tiaoxin-346 submitted to the CAESAR competition [5]. These schemes, not only rely on round-reduced AES (to process 16-byte message block, AEGIS-128L uses four rounds, while Tiaoxin-346 only three rounds of AES), but allow as well a full parallelization of the round calls. As a result, with AES-NI implementation they achieve exceptionally high efficiency and run at only 0.2–0.3 cycles per byte of message.

To understand the speed advantage of these designs, first we focus on AES-NI. We investigate the performance of the AES-NI instruction aesenc (executes one round of AES) on the latest Intel processors and deduce necessary conditions for efficient designs. Consequently, our designs have internal states composed of several 128-bit words (called blocks), while their step functions are based only on aesenc and bitwise additions (XORs). The state size, the number of aesenc calls per step, and the choice of state words to which aesenc is applied ensures that our designs will have a high efficiency.

Next, we focus on the security of the designs. The most common attacks for MACs and AE are internal collisions based on high probability differential characteristics that start and end in zero state differences (but some intermediate states contain differences, introduced through the messages). The inability of the adversary to efficiently built such collisions is the single security criteria required from our designs.

We consider two strategies that may lead to efficient and secure constructions. In the first, the AES rounds are applied to the words of the state in a way such that several steps of the construction mimic a few keyless AES rounds[1]. Due to the wide-trail approach, this strategy provides easier security proofs. However, we show that regardless of the step function chosen, such strategy has only limited efficiency potential. For instance, strategy based on 4-round AES can never run faster than 0.25 cycles per byte.

To achieve higher speed, we thus consider a second strategy, where message and state words can be XOR-ed between the AES calls. The wide-trail approach cannot longer be used (as each application is one-round AES), hence the security proof for the constructions becomes much harder. To solve it, for each candidate construction we transform the collision problem into a MILP problem, and find

[1] This approach was chosen in Tiaoxin-346, where 2-round AES is used.

the optimal solution which corresponds to the characteristic with the highest probability. The cases where such probability is too low correspond to secure constructions.

We search for suitable designs based on the second strategy by gradually increasing the state size and decreasing the number of AES rounds per step. In some cases, several constructions have the same efficiency but provide different security margin. We implement each construction on the latest Intel processors and check if the theoretical and actual cycle per byte count match. We list 7 secure constructions that provide a good tradeoff between state size and efficiency. The smallest has 6 words, and runs at 0.22 c/B on Haswell, and 0.188 c/B on Skylake. The most efficient has 12 words, and runs at 0.136 c/B on Haswell, and 0.125 c/B on Skylake. This construction uses only 2 AES rounds per one block of message, and thus it is five times faster than the AES.

2 Designs Based on the AES Round Function

2.1 The AES Round Function and the Instruction Set AES-NI

AES is the current block cipher standard and a well-studied cryptographic construction. As such, parts of AES are used in many crypto designs. The usage ranges from the utilization of the AES S-box in some hash functions, to application of the AES round function in stream ciphers, and employment of the whole AES in particular authenticated encryption schemes. The AES contains three different block ciphers, which only differs by their key sizes: in the remaining of this paper, we simply write AES to refer to the 128-bit key version AES-128.

From a software perspective, it may seem that partitioning of the AES can go up to the four basic round function operations: SubBytes, ShiftRows, MixColumns, and AddRoundKey. However, actual fast implementations of AES rely on the so-called AES-NI: a special set of instructions available on the latest processors, dedicated to efficiently executing *rounds of AES*.[2] As efficiency is our primary goal, we further focus on designs based on the instruction set of AES-NI. More precisely, we use only the processor instruction aesenc, which performs one regular (not the last) round of AES on an input state S with a subkey K:

$$\mathsf{aesenc}(S, K) = \mathsf{MixColumns}(\mathsf{ShiftRows}(\mathsf{SubBytes}(S))) \oplus K.$$

Let us recall the two notions related to the performance of a processor instruction, namely, *the latency* and *the reciprocal throughput* of an instruction. Informally, latency is defined as the number of clock cycles required to execute an instruction, whereas the reciprocal throughput (further called throughput) as

[2] In addition to the encryption and decryption rounds, AES-NI includes as well instructions that perform subkey generation and inverse MixColumns. Note that the four individual round operations can be realized as a composition of different instructions from AES-NI. However, such composition would have greatly reduced efficiency in comparison to the round calls.

the number of clock cycles required to wait before executing the same instruction. In Table 1 are given the performances of aesenc on the five latest Intel's processors. For instance, on Intel's Ivy Bridge family of processors aesenc has a latency of 8 and a throughput of 1. This means that aesenc needs 8 cycles to execute one AES round, and it can be called consecutively after 1 cycle.

Table 1. The latency and throughput of aesenc on the latest Intel's processors.

Processor	Latency	Throughput
Sandy Bridge	8	1
Ivy Bridge	8	1
Haswell	7	1
Broadwell	7	1
Skylake	4	1

Our design strategies target the five latest Intel's processors: Sandy and Ivy Bridge (collectively referred to as *bridge), Haswell and Broadwell (referred to as *well), and Skylake.

2.2 Efficiency

Our goal is to devise a strategy that results in designs based on aesenc that have a superior efficiency over the AES. Improvements in efficiency can come from two concrete approaches: reduction of the number of rounds per message block, and, parallelization of the aesenc calls. Let us take a closer look at the two approaches.

Reducing the Number of Rounds. The AES has 10 rounds[3], i.e. it uses 10 aesenc calls[4] to process a 16-byte message. Removing several rounds from the AES leads to a block cipher susceptible to practical attacks. This, however, does not imply that any design (not only a block cipher) should necessary use around 10 aesenc calls. In fact, a common approach based on the AES, is to design cryptographic primitives that use only four AES rounds to process 16-byte data.

The goal of our design is to use a minimal number of calls to aesenc. For this purpose, we define a metric, called a *rate* of design:

Definition 1 (Rate). *The rate ρ of a design is the number of AES rounds (calls to* aesenc*) used to process a 16-byte message.*

[3] Here, we simply use AES to refer to the AES-128.

[4] The last round in AES is different and it is executed with a call to the AES-NI instruction aesenclast, which has similar performance to aesenc.

For instance, AES-128 has a rate of 10, AES-256 has a rate of 14, AEGIS-128L has a rate of 4, and Tiaoxin-346 a rate of 3. Obviously, a smaller rate may lead to more efficient designs.[5]

Parallelizing the Round Calls. A large improvement in efficiency may come by switching from serial[6] to parallel calls to aesenc.

A design is based on serial calls to aesenc (or to any other instruction of that matter), if the following aesenc is called only after the previous aesenc has finished. In such designs, the latency and the number of calls to aesenc give an immediate bound on the required number of cycles. An example of a serial construction is the cipher block chaining (CBC) mode because it requires the output of processing the previous message block in order to process the next message block.[7] As AES-128 has 10 rounds, on Haswell (where aesenc has a latency of 7), the AES-CBC requires $10 \cdot 7 = 70$ cycles to process 16-byte plaintext (see Fig. 1): the first round (the first call to aesenc) starts at cycle 0 and completes at 7, the second starts at 7 and completes at 14, ..., the 10th starts at 63 and completes at 70. As a result, the construction runs at $70/16 = 4.375$ cycles/byte (or c/B for short).

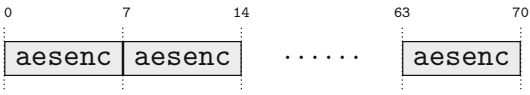

Fig. 1. Serial design: AES-CBC on Intel's Haswell with aesenc latency of 7 cycles. Only one message block is processed at once.

Designs with parallel calls to aesenc can be far more efficient, as the instructions are executed simultaneously, i.e. the following aesenc can be called while one or more of the previous aesenc are still executing. The cycle count now depends not only on the number of rounds and the latency, but also on the throughput and the maximal number of independent instances of aesenc supported by the design. A textbook example of parallelizable construction is the counter (CTR) mode.[8] On Haswell it is possible to process 7 message blocks in parallel (see Fig. 2): at cycle 0, aesenc is called and it will perform the first AES round for the first message block (and return the result at cycle 7); at cycle 1, aesenc for the first AES round of the second message block is called, etc., at cycle 6 the aesenc for the first round of the seventh message block is called. Then, aesenc that perform the second rounds for all the seven message blocks

[5] A smaller rate is not a sufficient condition of efficiency as parallelizing aesenc calls plays an important role as well (see the next paragraph).

[6] Bogdanov et al. [2] have analyzed the speed improvements of serial modes when processing multiple messages in parallel.

[7] Recall that the AES-CBC is defined as $C_{i+1} = \texttt{AES}_K(C_i \oplus M_{i+1})$.

[8] Recall that the AES-CTR is defined as $C_i = \texttt{AES}_K(N||i) \oplus M_i$, where N is a nonce.

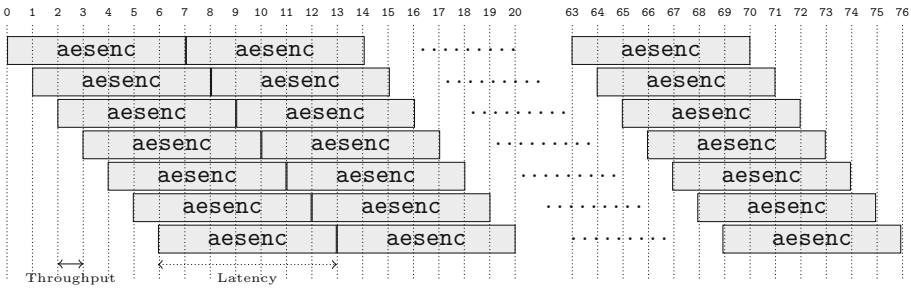

Fig. 2. Parallel design: `AES-CTR` on Intel's Haswell with `aesenc` latency of 7 cycles. It allows 7 message blocks to be processed in parallel. The `aesenc` is called every cycle.

are called at cycles 7–13. By repeating this procedure, it is possible to perform all ten `AES` rounds for all 7 message blocks – the last rounds are executed at cycles 63–69, and the ciphertexts are produced at cycles 70–76. Hence only 76 cycles, which can be brought down to 70 if longer messages are considered, are required to process 7 message blocks, or on average only 10 cycles per one message block (cf. to 70 cycles for processing a message block in the serial CBC mode). Therefore, the CTR mode runs at $10/16 = 0.625$ c/B, or precisely 7 times faster than the CBC mode.

The State Size and the Number of `aesenc` Calls per Step. The parallel calls to `aesenc` can be achieved only if the state size is sufficiently large. We have seen that CBC mode requires a state composed of only one 16-byte word, but provides no parallelization. On the other hand, if supplied with a state of seven words, the CTR mode can run seven instances in parallel. As we strive for designs with high efficiency and thus support for parallel calls to `aesenc`, they will have larger states. In general, *if the design makes c calls to `aesenc` per step, then the state has to have at least c 128-bit words*: only in this case we can have fully parallelizable `aesenc` calls.

The optimal number of `aesenc` calls per step depends on the latency to throughput ratio. *The most efficient designs use around latency/throughput independent calls to `aesenc` per one step.* Let us understand this fact on the example of a hypothetical design that has four `aesenc` calls per step to process 16-byte message (has a rate of $4/1 = 4$) and is implemented on Haswell, which in turn has a ratio of $7/1 = 7$. The four `aesenc` calls of the first step are called at cycles 0, 1, 2, and 3 (at every cycle because the throughput is 1), but the results of these calls are obtained only at cycles 7, 8, 9, 10 (because the latency is 7). As a result, at cycles 4, 5, and 6, no `aesenc` calls are made,[9] and we say that the `aesenc` port[10] has not been saturated, i.e. there have been empty cycles. Due

[9] Assuming that all the calls to `aesenc` of the next round depend on some of the outputs of the previous four `aesenc` calls.

[10] The part of the processor that executes `aesenc`.

to the empty cycles, even though the rate is 4, one needs 7 cycles on Haswell to process the message block, thus the speed is $7/16 = 0.4375$ c/B. The cycle count changes when the same design is implemented on Skylake (with ratio $4/1 = 4$). On this processor, the `aesenc` port is fully saturated, and on average it requires only 4 cycles per 16-byte message,[11] which means that this design would run at $4/16 = 0.25$ c/B.

A construction with rate ρ can run at most at 0.0625ρ c/B because, by definition, it needs ρ `aesenc` calls (in total at least ρ cycles) to process 16-byte message, hence the maximal speed is $\frac{\rho}{16} = 0.0625\rho$ c/B. On the other hand, if the number of `aesenc` calls per step is smaller than the latency to throughput ratio, then, for the aforementioned reasons, the `aesenc` port may not be saturated, and the speed may drop to $0.0625\frac{latency}{throughput}$ c/B. In the sequel, we take this number as our *expected speed*. The actual speed, however, may differ. It could be lower, if the `aesenc` between different steps are dependent, i.e. if the inputs to the `aesenc` of the next step depend on the outputs of the `aesenc` of the previous step. On the other hand, the actual speed could be higher than the expected, if more than $\frac{latency}{throughput}$ `aesenc` could run at the same time – this happens, when some of the `aesenc` calls of the next step can start before finishing most of the `aesenc` of the previous step.

Summary. Let us summarize the facts of this subsection as they provide hints to achieve high efficiency, i.e. low c/B measurement:

- lower rate (#`aesenc` per message block) leads to more efficient designs,
- all `aesenc` calls per step are independent and thus run in parallel,
- the state is at least as large as the number of `aesenc` calls per step,
- the #`aesenc` calls per step is close to the latency/throughput ratio.

2.3 Security Notions

We suggest design *strategies* to construct building blocks for symmetric-key primitives, and thus we adapt the security requirements accordingly. Our constructions proposed further, for instance, could be used to build a MAC algorithm, where an initialization phase first randomizes a 128-bit key and IV-dependent internal state to produce a 128-bit tag by injecting message blocks. In such a case, classical security requirements impose that no key-recovery or forgery succeeds in less than 2^{128} operations. If an authenticated encryption scheme uses our building block with a 128-bit key to produce a 128-bit tag, then as well, less than 2^{128} computations must not break the scheme.

Analyzing the resistance of a design against all possible attacks is infeasible without giving the full specification.[12] To capture this, we reduce the security

[11] If the `aesenc` are sufficiently independent between steps.

[12] For instance, the initialization and finalization stages of the constructed stream cipher or authenticated encryption scheme.

claim of our constructions to the problem of finding internal collisions. Nonetheless, we emphasize that this is only one of the requirements of a cryptographic primitive, thus the resistance against the remaining attacks should be checked after completing the whole design.

The reason we use state collisions as our unique security requirement is twofold. First, we cannot fathom how designers will use our building blocks, and this notion applies directly to many different schemes, like hash functions or MAC and AE where a state collision would yield forgery. Therefore, by focusing only on this notion, we maximize the security of future designs based on these building blocks. Second, the inherent algorithmic problem is well-studied and understood: it consists in finding special types of differential characteristics that start and end in zero difference. Finally, we can also argue how significant this requirement is by recalling that several primitives have been broken due to susceptibility to attacks based on state collisions (see for instance [13,20]).

To find a state collision means to identify two *different* sequences of messages such that, from the same initial state value, the same output state value is reached in the scheme after injecting the different message sequences. Consequently, we can describe this problem as finding a high-probability differential characteristic from the all-zero state difference to the same all-zero state difference, where the differences come from the message bytes. By high-probability, we mean higher than 2^{-128} since we focus on the AES, which relies on a 128-bit internal state.

To elaborate on the security reduction to state collisions, we briefly recall the wide-trail strategy adopted in the design of the AES [6]. This technique has been introduced to make the AES resistant to classical differential cryptanalysis in the single-key setting. In particular, the AES ensures a (tight) lower bound of the number of active S-boxes for any number of rounds in this model (see Table 2). In detail, the AES uses an Substitution-Permutation Network (SPN) including an MDS code to provably bound the diffusion, measured in terms of number of active S-boxes. Additionally, the S-box S from the substitution layer has been constructed to have a differential probability upper bounded by 2^{-6}, which means that any differential equation $\mathsf{S}(x) \oplus \mathsf{S}(x \oplus \delta_1) = \delta_2$ over $GF(2^8)$, for nonzero δ_1 and δ_2, has at most four solutions.

Table 2. Minimum number of active S-boxes in the AES in the single-key model.

Rounds	1	2	3	4	5	6	7	8	9	10
Active S-boxes	1	5	9	25	26	30	34	50	51	55

Therefore, to construct secure designs based on the AES round function when no differences are introduced in the subkeys, it is sufficient to ensure that a difference enters four rounds of AES. Indeed, four rounds necessarily have at least 25 active S-boxes, which directly yield an upper bound on any differential characteristic probability: $2^{-6 \cdot 25} = 2^{-150} \ll 2^{-128}$. This 4-round barrier explains why

many previous designs chose to exploit this provable bound and gain in efficiency in comparison to the ten rounds used in the actual AES-128 block cipher.

In our case, we are interested in designs which achieve higher performances and do not necessarily rely on four rounds of AES. Consequently, the differential characteristic mentioned before that starts and ends in no-difference states *must* activate at least 22 S-boxes, so its probability would be at most $2^{-6 \cdot 22} = 2^{-132} < 2^{-128}$. Hence, in the sequel the security goals imposed on our designs are such that their best differential characteristic has at least 22 active S-boxes.

2.4 General Structure and Definitions

We define here the classes of AES-based designs that we study in the remaining of the paper. For all the aforementioned reasons, we focus on only two operations on 128-bit values: the AES round function denoted by A and performed by the aesenc instruction, and the XOR operation denoted by $\oplus$.

More precisely, we study in Sect. 3 the class $\mathcal{A}^r_\oplus$ where the allowed operations on a state of s words belong to $\{A^r, \oplus\}$. The notation A^r refers to r cascaded iterations of the permutation A. Next, in Sect. 4, we move on to the more general class $\mathcal{A}^1_\oplus$ (simply denoted $\mathcal{A}_\oplus$), where the AES round function is not necessarily cascaded. The general structure of the elements of $\mathcal{A}_\oplus$ are depicted on Fig. 3, where we represent by dashed lines the optional components. We define an iteration of such designs as a *step* to avoid confusion with the *round function* A of the AES.

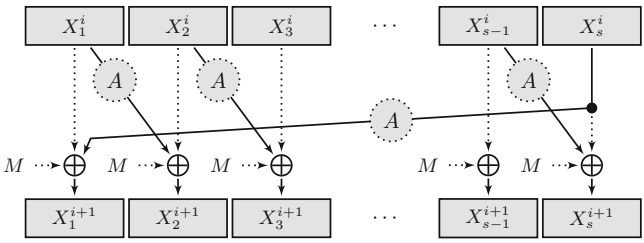

Fig. 3. One step of the general structure of the designs investigated in this paper. Dashed components mean they can be present or absent from the design.

We emphasize that all the designs belonging to these classes implement shifts of the state words to make the various applications of A to be *independent*. Consequently, each updated word X_t^{i+1}, for $0 \leq t < s$, necessarily depends on $X^i_{t-1 \pmod s}$, and optionally on X_t^i. The main rationale behind this stems from the objective to reach high efficiency: should the diffusion be higher, for instance where a single output of A would be XORed to every output words, the processor would have to wait until all the output words have their final value. In our case, the shifts allow to optimize the usage of the processor cycles: starting evaluating the design from right to left, the first call to A is likely to be finished evaluating

when we start processing the left-most state word. Hence, the iteration $i+1$ can start without waiting for the end of iteration i.

However, this optimized scheduling of instructions comes at the expense of the diffusion: from a single bit difference in the input state, reaching a full diffusion might take several steps. As a complete opposite, reaching full diffusion in a single step would mean XORing the output of a single A to all the output state words, and would waste many cycles. While this seems to suggest an interesting tradeoff, we nevertheless show in the sequel that there *do* exist designs in the class $\mathcal{A}_{\oplus}$ which, at the same time, achieve optimally high efficiency and meet our security requirements.

In terms of implementation, as mentioned before, the `aesenc` operations ends with the XOR of a round subkey and as a result, the implementations may benefit from this *free* operation. Namely, if we should XOR the message block M after the `aesenc`, we could just use the instruction `aesenc`$(\bullet, M)$. Otherwise, we might just use `aesenc`$(\bullet, 0)$.

Notations. We use the following notations to describe the designs. We introduce the parameters s that represents the number of 128-bit state words, a the number of `AES` rounds in a single step, and m the number of 128-bit message blocks processed per step. Additionally, we denote by ρ the rate of the design following Definition 1, that is $\rho = a/m$.

3 The Class $\mathcal{A}^r_{\oplus}$ and Rate Bounds

The class $\mathcal{A}^r_{\oplus}$, where $r > 1$, consists of designs that are based on r cascaded applications of the `AES` round function. This guarantees that state words will go through r rounds of `AES`, *without other state or message words being added to them*. Example of an actual construction from $\mathcal{A}^2_{\oplus}$ is given in Fig. 4. This design is based on 2-round `AES` as both of the words X^i_2 and X^i_5 will go through two `AES` rounds before any other state or message word is XORed to them.

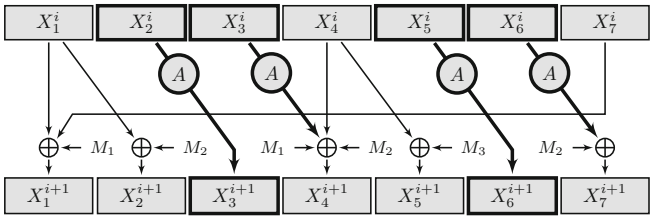

Fig. 4. A design from $\mathcal{A}^2_{\oplus}$.

Designs from $\mathcal{A}^r_{\oplus}$ are easier to analyze as they resemble r rounds of the `AES`. As a result, their main advantage lies in the possibility to use the wide-trail strategy of the `AES` which dictates that the minimal number of active S-boxes of

2, 3, and 4 rounds of AES is 5, 9, and 25 active S-boxes, respectively (see Table 2). For example, to prove that a particular $\mathcal{A}^3_\oplus$ design is secure by our definition, we have to show that in any differential characteristic that starts and ends in a zero difference, a state difference must go at least three times through the cascaded three rounds of AES. Such design would be secure, because the number of active S-boxes for any characteristic would be at least $3 \cdot 9 = 27 \geq 22$. For the class $\mathcal{A}^2_\oplus$ (resp. $A^4_\oplus$), the similar requirement is to activate five times (resp. once), the cascaded 2-round (resp. 4-round) AES.

The efficiencies of these designs, however, are limited. Further, we show that their rates cannot be arbitrary low, but are in fact bounded by r.

Theorem 1. *The rate ρ of a design based on $\mathcal{A}^r_\oplus$ cannot be less than r, i.e.*

$$\rho(\mathcal{A}^r_\oplus) \geq r.$$

Proof. Any design from $\mathcal{A}^r_\oplus$ can be divided into several parts. Each r-step cascaded aesenc with the corresponding state words composes a so-called nonlinear part. Consecutive XORs of the message and the state words (with no aesenc in between) also compose a part, called a linear part. Note, there can be several nonlinear and linear parts. For instance, the design from Fig. 4 can be divided into two nonlinear parts (denoted with thick lines) and two linear parts (the remaining two parts between the nonlinear parts).

A design is insecure if we can build a high-probability differential characteristic that starts and ends in zero state difference (but some intermediate state words have non-zero differences introduced through the message words). Further, we show that if the rate is too small, more precisely if $\rho < r$, then we can build a differential characteristic with no active S-boxes. That is, the difference in the state can be introduced through the message words and then canceled in the following steps, without reaching the state words to which aesenc is applied. As a result, the probability of that differential characteristic would be one.

Let m be the number of message blocks XORed per step. Moreover, let N and L be the total number of nonlinear and linear parts, respectively. Recall that each of the N nonlinear parts has at least r cascaded applications of aesenc. Thus, for the rate ρ, defined as the number of aesenc calls per message block, it holds:

$$\rho(\mathcal{A}^r_\oplus) \geq \frac{N \cdot r}{m}. \tag{1}$$

To build a differential characteristic with no active S-boxes, at each step of the characteristic the difference that enters each of the N nonlinear parts should be zero. This condition can be expressed as a system of linear equations where the differences in the message words are the unknown variables. At each step, we require the inputs to the nonlinear parts to be zero. Hence, for each step, N equations are added to the system, and the number of variables is increased by m. For instance, the system that corresponds to the design from Fig. 4 has the following four equations that correspond to the first two steps of the characteristics for the two non-linear layers:

$$\begin{aligned} \Delta M_2^1 &= 0, \\ \Delta M_3^1 &= 0, \\ \Delta M_1^1 \oplus \Delta M_3^2 &= 0, \\ \Delta M_1^1 \oplus \Delta M_2^1 \oplus \Delta M_3^2 &= 0, \end{aligned}$$

where the unknown ΔM_i^j is the difference in the message word M_i at step j of the characteristic.

The resulting system is homogeneous because we require the input differences to the nonlinear layers to be zero. When built for a differential characteristic on R steps, the system has $m \cdot R$ variables. Furthermore, it has $N \cdot R$ equations that correspond to the conditions that zero differences enter all nonlinear layers, and additional s equations (where s is the number of state words) that correspond to the conditions that all state words after step R have a zero difference. As the system is homogeneous, it has a non-zero solution as long as the number of variables exceeds the number of equations, i.e. $m \cdot R > N \cdot R + s$, or equivalently, as long as

$$R(m - N) - s > 0. \tag{2}$$

We show that if $\rho < r$, then (2) holds. From (1), it follows that $r > \rho \geq \frac{N \cdot r}{m}$, hence $N < m$. Let $m - N = t > 0$. As the number s of state words is fixed, and the number of steps R of the differential characteristic can increase, it follows that $R(m - N) - s = R \cdot t - s > 0$, when $R > \frac{s}{t}$. Therefore, when the rate ρ of the design is smaller than r, the homogeneous system has a non-zero solution which corresponds to a differential characteristic with no active S-boxes and, as a result, the design is insecure. Hence, the rate ρ of a secure design cannot be less than r. □

Remark 1. The rate bound holds for any design based on r-round cascaded AES (and not only for the class with shifts to the right, that we analyze).

From the theorem, we can conclude that regardless of the actual construction, designs from $\mathcal{A}_{\oplus}^4, \mathcal{A}_{\oplus}^3$ and $\mathcal{A}_{\oplus}^2$ cannot have rates lower than 4, 3, and 2, respectively, and thus cannot run faster than 0.250 c/B, 0.188 c/B, and 0.125 c/B, respectively.

Note, as the step functions of AEGIS-128L and Tiaoxin-346 run at 0.250 c/B and 0.188 c/B (have rates 4 and 3), in order to find more efficient designs, we have to either find rate-3 designs with smaller states (at most 12 words as Tiaoxin-346 has 13 words), or designs with lower rate. We have run a complete search of all designs from $\mathcal{A}_{\oplus}^3$ with at most 12 state words and found that none of them is secure[13]. Furthermore, we have run a partial search[14] among designs from $\mathcal{A}_{\oplus}^2$ and found constructions with rate 2.66, but not lower. Thus, to achieve more efficient designs, in the next section we examine the class $\mathcal{A}_{\oplus}$.

[13] This gives a rise to the conjecture that the inequality from the theorem is strict.
[14] In this case, the search space cannot be exhausted as it is too large.

4 Designs in the Class $\mathcal{A}_{\oplus}$

In this section, we focus on the more general class of designs $\mathcal{A}_{\oplus}$, where the AES round function is not necessarily iterated. From a cryptanalytic standpoint, it means this class encompasses designs where state differences can be introduced between two consecutive AES round functions. The main consequence in comparison to the previous class $\mathcal{A}^r_{\oplus}$ from Sect. 3 is that we lose the simplicity of the analysis brought by the wide-trail strategy. One could compare the change of analysis as transition from the single-key framework of the AES to its related-key counterpart (where differences may be introduced between consecutive rounds).

However, in spite of the more complex analysis, we show there exists low-rate designs in this larger class that meet our security requirements. Namely, we show several designs that achieve rates 3, 2.5, and even rate 2.

The study of $\mathcal{A}_{\oplus}$ is less straightforward than the previous case, thus we rely on mixed integer linear programming (MILP) to derive lower bounds on the number of active S-boxes the designs. In the next sections, we briefly recall the MILP technique applied to cryptanalysis (Sect. 4.1) and we detail our results (Sect. 4.2).

4.1 MILP and Differential Characteristic Search

From a high-level perspective, a MILP problem aims at optimizing a linear objective function subject to linear equalities and/or linear inequalities. The technique we use in this paper is said to be *mixed* integer linear programming as it alleviates the all-integer constraint on the classical linear programming variables. More precisely, in our case some variables might not be integers, but all the integer variables are 0–1 variables. Therefore, we could dub this particular setup as 0–1 MILP.

The 0–1 MILP problems are usually NP-hard, but solutions can be found using different strategies, for instance, the cutting-plane method which iteratively refines a valid solution by performing cuts relying on the linear inequality constraints of the problem. For our purposes, we use one of the many solvers existing to date, namely the Gurobi solver [10]. Several published results rely on MILP optimization tools to solve cryptanalytic problems: searches for differential characteristics in various schemes are given in [18], known lower bounds for the number of active S-boxes for the related-key setting of AES in [16], analysis of reduced versions of the Trivium stream cipher in [4], etc.

We aim at finding differential characteristics from the all-zero difference input state to the same all-zero output state after a variable number of steps. As mentioned before, our measure of security relies on the number of active S-boxes, which gives an upper bound on the success probability of a differential attack that may lead to state collisions. We transform the search of differential characteristics into MILP problems whose objective functions count (and minimize) the number of active S-boxes. In practice, since we use the AES round function, we only require the differential characteristics to have *at least* 22 active S-boxes to ensure security.

Let $\mathcal{X} = \{x_i, i = 1, \ldots, m\}$ be the set of all the m variables and $\mathcal{S} \subseteq \mathcal{X}$ be the subset of variables representing the S-boxes of the scheme. With these notations, a classical MILP problem that we study can be stated as follows:

$$\begin{aligned} \text{Minimize: } & \sum_{x \in \mathcal{S}} x, \\ \text{subject to: } & \mathbf{A}x = b, \quad x_i \text{ all 0–1 variables.} \end{aligned}$$

Recall that, for each $x \in \mathcal{S}$, $x = 1$ if and if only if the S-box associated to x is active. The other variables in $\mathcal{X} \setminus \mathcal{S}$ represent intermediate state differences.

For a given state size of s 128-bit words, to express the problem of finding a differential characteristic, we examine the effect of the four elementary transformations of the AES round function. We emphasize that the analysis is performed in terms of truncated differences ($x \in \{0, 1\}$) since we are only concerned about active or inactive S-boxes: the actual differences are insignificant. Therefore, as soon as one S-box is active, the SubBytes operation maintains this property. Hence, SubBytes does not introduce any linear constraints in the MILP problem. The same holds for the ShiftRows operation, which only permutes the bytes of the internal state.

However, the MixColumns operation implements a linear code with maximal distance (MDS), and it does introduce linear constraints in the MILP problem. Namely, the new inequalities enforce the minimal distance into the problem description. Assuming that the MixColumns operation is applied to the variables (representing truncated differences) $[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$ to produce $[x'_i, x'_{i+1}, x'_{i+2}, x'_{i+3}]$, we introduce the nine following inequalities:

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 + x'_1 + x'_2 + x'_3 + x'_4 - 5t \geq 0, \\ t - v \geq 0, \quad v \in \{x_1, x_2, x_3, x_4, x'_1, x'_2, x'_3, x'_4\}. \end{aligned}$$

The usage of the extra temporary variable t ensures that the MDS bound is valid as soon as one of the x variables is nonzero (i.e. zero or at least five variables equal one).

Finally, the AddRoundKey operation XORs a 128-bit subkey into the state, which also introduces linear inequalities in the MILP problem description. Consider the XOR $y = x_1 \oplus x_2$ of two variables $x_1, x_2 \in \{0, 1\}$ representing two truncated differences. In the event that $(x_1, x_2) = (0, 0)$, y naturally becomes 0, and y becomes 1 if $(x_1, x_2) \in \{(0, 1), (1, 0)\}$. However, the behavior is undetermined when $(x_1, x_2) = (1, 1)$ as y can either be 0 or 1 depending on the *actual* values of the corresponding differences. Indeed, because we lose information by compressing the differences to truncated differences, we lose the information on the possible equality of differences. Consequently, we have to consider both cases: $y \in \{0, 1\}$. This partial behavior of XOR is captured by the four following inequalities:

$$\begin{aligned} x_1 + x_2 + y - 2t \geq 0, \\ t - v \geq 0, \quad v \in \{x_1, x_2, y\}, \end{aligned}$$

which basically excludes the case where only one of the three variables equals one.

In summary, for a single round of `AES`, we introduce $4 \times 9 + 16 \times 4 = 100$ inequalities to express the round constraints. On top of that, we introduce $16 \times 4 = 64$ additional inequalities for every extra XORs required to inject the message blocks. Finally, we also need to add $2 \times s \times 16$ equality constraints to represent the required zero difference in the input state and in the output state to reach a state collision. To give concrete numbers, we point out that systems corresponding to our smaller designs would need around 10,000 binary variables and 20,000 to 30,000 linear constraints.

Limitations. Despite providing a simple and efficient way of finding differential characteristics, MILP only yields upper bounds on the actual probabilities of the differential characteristics as, theoretically, they can be impossible. We emphasize that this does not relate to impossible differential characteristic, but to the fact that partially undetermined behavior of the XOR operation (mentioned before) may result in inconsistent systems that produce truncated differential characteristics which are impossible to instantiate with actual differences. Fortunately, while a cryptanalyst should ensure the validity of the produced characteristics, we, as designers, only need to confirm that the upper bound on the probability of the best differential characteristic is sufficiently low.

4.2 Results of the Search

In this section, we conduct the search for efficient designs and describe the results produced by the MILP analysis. In the next Sect. 5, we give the actual implementations and benchmarks of the produced designs.

Rate 3. We start the search with rate-3 designs and try to minimize the number s of state words. For a given state size s, the general structure depicted in Fig. 3 contains at most 12^s different designs. As the smallest possible size is $s = 3$, we efficiently exhaust all the 12^3 designs. In this reduced space, we have found that not a single design can reach 22 active S-boxes. Furthermore, for $s = 4$ state words, there exists secure constructions, albeit incompletely saturating the `aesenc` port for all the current processors[15], thus we do not consider them.

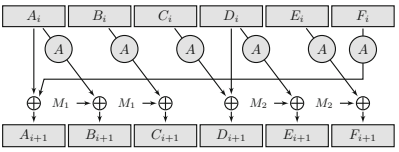

Fig. 5. Rate-3 design with 6 words.

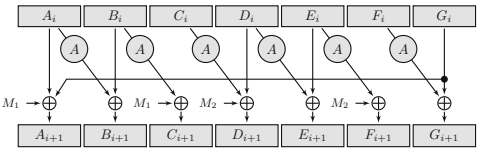

Fig. 6. Rate-3 design with 7 words.

[15] The design uses only 3 `aesenc` calls per round, whereas the smallest latency among all the processors is 4.

Having this objective in mind for rate-3 designs, we then move to step functions having either six calls to A (this saturates the `Skylake aesenc` port as `aesenc` has latency of 4) and inject two message blocks in each step, or nine calls to A (to saturate `*bridge` and `*well aesenc` ports) and inject three blocks. We find three different designs with state sizes of 6, 7 and 8 words, respectively, that are best suitable for `Skylake`. These designs achieve different security margins with lower bounds of 22, 25 and 34 active S-boxes, respectively (refer to Figs. 5, 6 and 7).

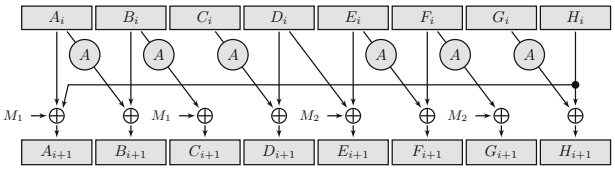

Fig. 7. Rate-3 design with 8 words.

For the case of nine calls to A (suitable as well for `*bridge` and `*well` architectures), we propose the design from Fig. 8 that reaches a minimum of 25 active S-boxes, has nine state words, and uses no additional XOR operations.

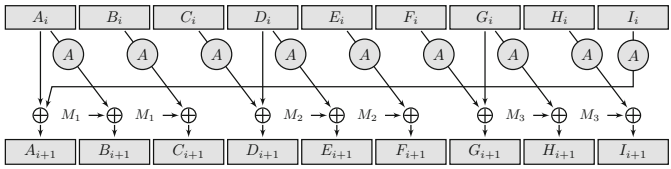

Fig. 8. Rate-3 design with 9 words.

Rate Smaller Than 3. To reach rates smaller than three, we first consider cases with two message blocks injected in every step. This restricts the number of A per round to five. We have performed a search within these restrictions and found constructions with seven and eight state words (see Figs. 9 and 10). The two design achieves rate $5/2 = 2.5$, have at least 22 and 23 active S-boxes, and saturate the `aesenc` port on `Skylake` processor.

Finally, we consider designs with rate of 2. We have not found a construction that injects two message blocks per step, however, we have discovered one that processes three message blocks per step (see Fig. 11). It has 12 state words, uses 6 `aesenc` to process 3 message blocks, and has at least 25 active S-boxes in any differential characteristic. Note, this construction compares very favorably to `Tiaoxin-346`: it is more compact and more efficient at the same time, since `Tiaoxin-346` reaches rate 3 with 13 state words.

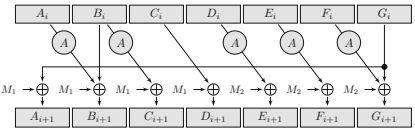

Fig. 9. Rate 2.5 with 7 state words.

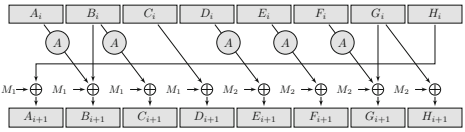

Fig. 10. Rate 2.5 with 8 state words.

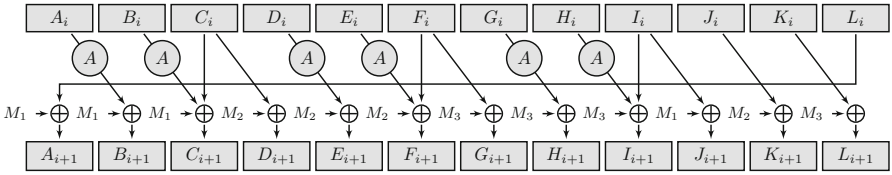

Fig. 11. Rate-2 design with 12 state words.

5 Implementations Results

We benchmark the seven constructions on the latest Intel's processors. The `aesenc` on some of these processors have similar performances (see Table 1), thus we benchmark on only three different platforms: Ivy Bridge (i5-3470) with Linux kernel 3.11.0-12 and gcc 4.8.1, Haswell (i5-4570) with Linux kernel 3.11.0-12 and gcc 4.8.1, and Skylake (i5-6200U) with Linux kernel 3.16.0-38 and gcc 4.8.4. We wrote the implementations in `C` and optimized them separately for each processor. The benchmarks were produced with disabled Turbo Boost and for 64kB messages[16].

The produced benchmarks are given in Table 3. Recall that our expected speed (expressed in c/B) is defined as $0.0625 \cdot \max(\rho, \frac{latency}{throughput})$. When the measured speed matches the expected (at most 5 % discrepancy), in the table we give the expected speed in **bold text**. On the other hand, when the measured speed is lower (resp. higher) than the expected, we give the actual speed with superscript $-$ (resp. $+$).

From the table, we can see that in most of the cases, our benchmarks follow the expected speed. For Ivy Bridge, the exceptions are the rate-3 design, which runs in 0.222 c/B instead of the expected 0.189 c/B (17 % slower), and the rate-2 design that runs at 0.190 c/B instead of 0.167 c/B (13 % slower). For Haswell, three designs run faster than expected, with gains of 15 %, 24 %, 22 %, respectively. On Skylake, the measured speed matches the expected speed for all seven constructions.

Among the seven constructions, we would like to single out the last constructions that has rate of 2, i.e. it uses two `AES` rounds to process a 16-byte message. On all of the three tested processors, this construction is exceptionally efficient. In addition, on Skylake, we were able to match the actual theoretical speed (our

[16] Only a slight degradation of speed is observed when the message length is a few kilobytes.

Table 3. Benchmarks (in c/B) of designs based on the AES round function. s: number of 128-bit state words, a: number of AES rounds in a single step, m: number of 128-bit message blocks processed per step, x number of additional XORs per step, ρ: rate of design (a/m), LB: lower bound on the number of active S-Boxes. Highlighted numbers means that the aesenc port is saturated for the given processor. Numbers in parentheses are projections, no actual measurements have been performed. Numbers in bold denotes that practical and theoretical speed match (less than 5 % difference), while numbers with + (resp. −) denote that the practical speed is higher (resp. lower) than the theoretical.

s	a	m	x	ρ	LB	Speed in c/B			Reference
						*bridge	*well	Skylake	
5	5	1	1	5	25	(0.500)	(0.436)	(0.313)	AEGIS-128 [19]
6	6	1	1	6	25	(0.500)	(0.436)	(0.375)	AEGIS-256 [19]
8	8	2	2	4	25	(0.250)	(0.250)	(0.250)	AEGIS-128L [19]
13	6	2	4	3	30	(0.250)	(0.219)	(0.188)	Tiaoxin-346 [17]
6	6	2	0	3	22	**0.250**	**0.219**	**0.188**	Figure 5
7	6	2	3	3	25	**0.250**	**0.219**	**0.188**	Figure 6
8	6	2	4	3	34	**0.250**	**0.219**	**0.188**	Figure 7
9	9	3	0	3	25	0.222^{-}	**0.188**	**0.188**	Figure 8
7	5	2	4	2.5	22	**0.250**	0.189^{+}	**0.156**	Figure 9
8	5	2	5	2.5	23	**0.250**	0.177^{+}	**0.156**	Figure 10
12	6	3	9	2	28	0.190^{-}	0.136^{+}	**0.125**	Figure 11

measured speed was 0.126 c/B against the theoretical 0.125 c/B). Hence, designs based on this construction may run five times faster than AES-128.

We note that on platforms without AES-NI support our design cannot reach the target speed. However, by no means they are slow as they use only 2–3 AES rounds to process 16-byte message block. Hence, the expected speed on these platforms is still much higher than the speed of AES, e.g. we expect that our constructions will run around 3–5 times faster than AES-128 in counter mode.

In addition, the state sizes of the constructions are large hence they are not suitable for lightweight applications. However, we note that all seven constructions have sizes which are smaller than the state of SHA-3 which has 25 64-bit state words (equivalent to 12.5 128-bit blocks).

6 Conclusion

We have presented new building blocks for secret-key primitives based on the AES round function. By targeting the most recent Intel processors from the past

four years, we have relied on the dedicated instruction set `AES-NI` to construct highly efficient designs. The designs are finely tuned for these processors to take advantage of the available parallelism and to reach optimal speed. They are based on the second, more efficient design strategy which requires a more complex security proof (reduction to MILP), but allows higher efficiency.

We have provided seven different building blocks that follow our design strategies and that reach high speed on the latest processors. On Ivy Bridge they run at 0.190–0.250 c/B, on Haswell at 0.136–0.219 c/B, while on Skylake at 0.125–0.188 c/B. We emphasize that our fastest construction uses only two `AES` rounds to process 16-byte message and on Skylake runs at only 0.125 c/B. To the best of our knowledge, this construction is much faster than any known cryptographic primitive.

Follow-up works to introduce better designs may start from two related directions: either by trying to reduce the state size, or by increasing the number of processed message in each step of the designs. The former might be useful to improve so designs that requires too many registers and slow down the whole process. The latter would automatically reduce the rate of the design and directly affect the measured speed. This direction is however difficult to tackle as the adversary has a lot more freedom to construct high-probability characteristics.

References

1. Biryukov, A.: The design of a stream cipher LEX. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 67–75. Springer, Heidelberg (2007)
2. Bogdanov, A., Lauridsen, M.M., Tischhauser, E.: Comb to pipeline: fast software encryption revisited. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 150–171. Springer, Heidelberg (2015)
3. Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., Tischhauser, E.: ALE: AES-based lightweight authenticated encryption. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 447–466. Springer, Heidelberg (2014)
4. Borghoff, J., Knudsen, L.R., Stolpe, M.: Bivium as a mixed-integer linear programming problem. In: Parker, M.G. (ed.) Cryptography and Coding 2009. LNCS, vol. 5921, pp. 133–152. Springer, Heidelberg (2009)
5. CAESAR. Competition for Authenticated Encryption: Security, Applicability, and Robustness. http://competitions.cr.yp.to/caesar.html
6. Daemen, J., Rijmen, V.: The Design of Rijndael: ALE - The Advanced Encryption Standard. Springer, Heidelberg (2002)
7. Daemen, J., Rijmen, V.: A new MAC construction ALRED and a specific instance ALPHA-MAC. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 1–17. Springer, Heidelberg (2005)
8. Daemen, J., Rijmen, V.: The MAC function Pelican 2.0. Cryptology ePrint Archive, report 2005/088 (2005)
9. Derbez, P., Fouque, P.-A., Jean, J.: Improved key recovery attacks on reduced-round `AES` in the single-key setting. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 371–387. Springer, Heidelberg (2013)
10. Gurobi Optimization, Inc.: Gurobi Optimizer Reference Manual (2015)

11. Jakimoski, G., Khajuria, S.: ASC-1: an authenticated encryption stream cipher. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 356–372. Springer, Heidelberg (2012)
12. Käsper, E., Schwabe, P.: Faster and timing-attack resistant AES-GCM. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 1–17. Springer, Heidelberg (2009)
13. Khovratovich, D., Rechberger, C.: The LOCAL attack: cryptanalysis of the authenticated encryption scheme ALE. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 174–184. Springer, Heidelberg (2014)
14. Li, L., Jia, K., Wang, X.: Improved single-key attacks on 9-round AES-192/256. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 127–146. Springer, Heidelberg (2015)
15. Mala, H., Dakhilalian, M., Rijmen, V., Modarres-Hashemi, M.: Improved Impossible differential cryptanalysis of 7-round AES-128. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 282–291. Springer, Heidelberg (2010)
16. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Wu, C.-K., Yung, M., Lin, D. (eds.) Inscrypt 2011. LNCS, vol. 7537, pp. 57–76. Springer, Heidelberg (2012)
17. Nikolić, I.: Tiaoxin-346. Submission to the CAESAR Competition (2014)
18. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 158–178. Springer, Heidelberg (2014)
19. Wu, H., Preneel, B.: AEGIS: a fast authenticated encryption algorithm. Cryptology ePrint Archive, report 2013/695 (2013)
20. Wu, S., Wu, H., Huang, T., Wang, M., Wu, W.: Leaked-state-forgery attack against the authenticated encryption algorithm ALE. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 377–404. Springer, Heidelberg (2013)