A Low-Area ASIC Implementation of AEGIS128 – a Fast Authenticated Encryption Algorithm

Robert Schilling, Manuel Jelinek, Markus Ortoff, and Thomas Unterluggauer
Graz University of Technology
Institute for Applied Information Processing and Communications
Email: {rschilling, jelinek, ortoff}@student.tugraz.at, thomas.unterluggauer@iaik.tugraz.at

***Abstract*—Due to the lack of proper dedicated authenticated encryption algorithms, the CAESAR cryptographic competition aims to find new such algorithms. The goal of authenticated encryption is to provide both confidentiality and authenticity within a single algorithm. This paper introduces the first application-specific integrated circuit of AEGIS128, which is one promising submission to the CAESAR competition. The dedicated hardware design is optimized towards yielding the smallest area for AEGIS128. Using a 0.13 µm low-leakage process from Faraday Technology, the design requires merely 13,558 gate equivalents or 0.06942 mm^2. Simulations of this design at a clock frequency of 100 MHz result in 65 Mbps data throughput.**

I. Introduction

Confidentiality and authenticity are both fundamental services in cryptography. Especially when sending data over an insecure channel, it is desirable to guarantee data authenticity in addition to confidentiality. An algorithm which provides such functionality is called an Authenticated Encryption (AE) algorithm.

One method to provide authenticated encryption is to use a block cipher such as the Advanced Encryption Standard (AES) [6] in different modes of operation. Confidentiality alone is provided using, e.g., AES in the Cipher Block Chaining mode (CBC) [7]. For authenticated encryption, an additional, separate authentication step is needed in order to generate the Message Authentication Code (MAC) that ensures the authenticity. This can be done using a cipher-based MAC (CMAC) [8] algorithm. In such a setup, a second key to provide the authenticity is needed in addition to the encryption key. More efficient than providing confidentiality and authenticity separately are block cipher modes of operation for authenticated encryption, e.g., AES-CCM (AES Counter with CBC-MAC) [10], or AES-GCM (AES Galois/Counter Mode) [9].

To avoid the use of a full block cipher and the overhead of two separate processing steps and keys, a dedicated authenticated encryption algorithm is desirable. Due to the lack of proper algorithms, the international cryptographic research community decided to start a competition called CAESAR [2], which stands for "Competition of Authenticated Encryption: Security, Applicability and Robustness". This challenge attempts to find suitable authenticated encryption algorithms that have advantages over the existing block cipher modes of operation for authenticated encryption. For comparing the submissions and for proper selection of the final algorithms, optimized implementations are needed. Since there are no published hardware implementations yet, we show the first area-optimized Application-Specific Integrated Circuit (ASIC) design of AEGIS128 [12], which is one of the candidates to CAESAR.

The suite of AEGIS algorithms provides three different variants, respectively AEGIS128, AEGIS256 and AEGIS128L. These algorithms differ in key size, state size and performance. This paper describes an ASIC implementation of AEGIS128 that is optimized for low area. The hardware design results in an area of 13,558 Gate Equivalents (GE). When operating at a clock frequency of 100 MHz, a data throughput of 65 Mbps is reached. In addition to the ASIC implementation, we present the synthesis results on a Xilinx Artix-7 Field-Programmable Gate Array (FPGA).

The paper is structured as follows: in Section 2 the AEGIS128 algorithm is introduced. Section 3 and 4 describe the ASIC implementation of AEGIS128 and how to communicate with the host interface. Section 5 describes which optimization techniques were used to reduce the chip size. Section 6 details the implementation of AEGIS128 on an Artix-7 FPGA. Finally, Section 7 presents the implementation results.

II. Algorithm

This section intends to give a short introduction to the construction of AEGIS128 [12]. AEGIS128 uses a 128-bit key, a 128-bit initial vector and a 640-bit state to encrypt and authenticate 128-bit message blocks. In addition, the algorithm also provides authenticity for dedicated associated data, which is, e.g., non-confidential metadata.

The core function of this algorithm is *StateUpdate128*, which is based on the AES round function [6], denoted *AES-Round*. The AES round function uses the following subfunctions: *AES-AddRoundKey*, *AES-ShiftRows*, *AES-MixColumns* and *AES-SubBytes*. *StateUpdate128* updates the state S_i, which is split into five 128-bit parts $S_{i,j}$ (j=0,..,4), with a 128-bit message block m_i as follows:

```
function StateUpdate128 $(S_i, m_i)$
    $S_{i+1,0} \leftarrow$ AESRound$(S_{i,4}, S_{i,0} \oplus m_i)$
    $S_{i+1,1} \leftarrow$ AESRound$(S_{i,0}, S_{i,1})$
    $S_{i+1,2} \leftarrow$ AESRound$(S_{i,1}, S_{i,2})$
    $S_{i+1,3} \leftarrow$ AESRound$(S_{i,2}, S_{i,3})$
    $S_{i+1,4} \leftarrow$ AESRound$(S_{i,3}, S_{i,4})$
    return $S_{i+1}$
```

978-1-4799-7244-9/14/$31.00 © 2014 IEEE

The first step is to load the 128-bit key K_{128} and the 128-bit initialization vector IV_{128} to assemble the initial state S_{-10}. As specified in [12], the initialization also uses two constants: $const0$ is defined as `0x6279E990593722150D08050302010100` and $const1$ is defined as `0xDD28B57342311120F12FC26D55183DDB`.

After initialization, the state is updated 10 times using *StateUpdate128*. Thereby, the input to *StateUpdate128* is alternated between K_{128} and IV_{128}.

```
$S_{-10,0} \leftarrow K_{128} \oplus IV_{128}$
$S_{-10,1} \leftarrow const1$
$S_{-10,2} \leftarrow const0$
$S_{-10,3} \leftarrow K_{128} \oplus const0$
$S_{-10,4} \leftarrow K_{128} \oplus const1$
for $i = -5$ to $i = -1$ do
    $m_{2i} \leftarrow K_{128}$
    $m_{2i+1} \leftarrow IV_{128}$
for $i = -10$ to $i = -1$ do
    $S_{i+1} \leftarrow StateUpdate128(S_i, m_i)$
```

For associated data, the algorithm ensures authenticity without confidentiality. If associated data *AD* is provided, i.e., its length *adlen* > 0, the state is updated in the next step by invoking *StateUpdate128* with all associated data. Thereby, 128 bit of associated data are processed per update round. This happens u_L times whereas u_L is $\lceil \frac{adlen}{128} \rceil$. If no associated data is provided, this processing step is skipped.

$$S_{i+1} \leftarrow StateUpdate128(S_i, AD_i)$$

Next, each 128-bit block of the plaintext P_i is encrypted to the ciphertext block C_i by XORing P_i with parts of the current state S_{u_L+i}. In the respective computation below, & denotes a bitwise AND operation. After the encryption of each block, the state is updated using *StateUpdate128* by using the plaintext block P_i as the input. This procedure is repeated v_L times whereas v_L is defined as $\lceil \frac{msglen}{128} \rceil$.

$$C_i \leftarrow P_i \oplus S_{u_L+i,1} \oplus S_{u_L+i,4} \oplus (S_{u_L+i,2} \& S_{u_L+i,3})$$
$$S_{u_L+i+1} \leftarrow StateUpdate128(S_{u_L+i}, P_i)$$

The last step is to compute the MAC, which is also called the tag T. Therefore, a temporary 128-bit field tmp that is assembled from the two 64-bit integers $adlen$ (length of associated data) and $msglen$ (message length) is XORed with the current state $S_{u_L+v_L,3}$. Using tmp as an input, *StateUpdate128* is then applied to the state seven times. Finally, the tag T is computed by XORing all 128-bit parts $S_{u_L+v_L+7,i}$ of the final state $S_{u_L+v_L+7}$.

```
$tmp \leftarrow S_{u_L+v_L,3} \oplus (adlen || msglen)$
for $i = u_L + v_L,$ to $i = u_L + v_L, +6$ do
    $S_{i+1} \leftarrow StateUpdate128(S_i, tmp)$
$T \leftarrow \oplus_{i=0}^{4} S_{u_L+v_L+7,i}$
```

Ciphertext decryption works almost the same as encryption: the resulting ciphertext C is loaded as the message and the same processing as during encryption is done. However, it is necessary to perform the state updates using the decrypted plaintext that is produced during the computation. Moreover, it is important to only use valid bits from the decrypted message P. This needs to be taken into account for the last block if the message length *msglen* is not a multiple of 128 bit. All bits exceeding the actual message length have to be set to zero.

$$P_i \leftarrow C_i \oplus S_{u_L+i,1} \oplus S_{u_L+i,6} \oplus (S_{u_L+i,2} \& S_{u_L+i,3})$$
$$S_{u_L+i,1} \leftarrow StateUpdate128(S_{u_L+i}, P_i)$$

The tag T required for verifying the authenticity of the message is finally computed the same way as during encryption.

III. Implementation

The hardware implementation splits the design up into three major components: first, an 8-bit *AMBA Peripheral Bus Slave* [1], second, the *datapath*, and third, the *control unit*. The AMBA Peripheral Bus (APB) is used for communication with the host. The interaction between these modules is shown in Fig. 1.

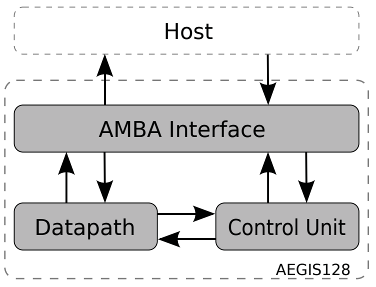

Fig. 1. Top-level structure of the AEGIS128 chip.

The datapath is controlled by the control unit, which is implemented as a Finite State Machine (FSM). The FSM sets its control signals depending on the current state and therefore triggers an action in the datapath. The latter implements all required data operations and therefore contains appropriate combinatorial logic and registers.

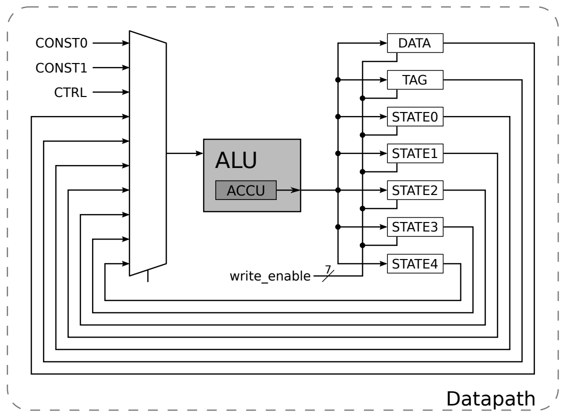

Fig. 2. Datapath structure of AEGIS128 - all paths have a width of 128 bit.

The datapath shown in Fig. 2 contains an Arithmetic Logic Unit (ALU), a 640-bit state register `STATE`, and two shared registers: `DATA` (128 bit, also used for the *key*) and `TAG` (128 bit, also used for the intialization vector *iv*, the assosciated data length *adlen*, and the message length *msglen*). A 128-bit multiplexer structure is responsible for routing the result of the ALU depending on the current operation.

The ALU is built around a 128-bit accumulator and provides six different operations. The operations are: `Load ACCU`, `XOR`, `AND`, `AES-ShiftRows`, `AES-MixColumns` and `AES-SubBytes`. Compared to a standard AES implementation, AEGIS does not need the inverse functions of `AES-MixColumns`, `AES-ShiftRows` and `AES-SubBytes`. The operation `AES-SubBytes` substitutes single bytes according to the AES S-Box and contains one pipeline stage. This means it takes 17 cycles to perform `AES-SubBytes` operations on the whole accumulator: one cycle for filling the pipeline and 16 cycles to perform the operation on all bytes in the accumulator. `AES-MixColumns` operates on four bytes per cycle, which means it needs to be executed four times to perform the operation on the whole accumulator. The input vector of the ALU defines on which bytes the operation should be done. All other operations work on the whole accumulator in one clock cycle. To save area, the S-Box of `AES-SubBytes` and `AES-MixColumns` are implemented as optimized versions, which will be described in the following.

A. *AES-MixColumns*

AES-MixColumns is based on a multiplication operation in the Galois Field GF(2^8). To avoid using a dedicated multiplier, the implementation is done using left shift operations (with a conditional addition of the irreducible polynomial `0x1b`) and additions as proposed in [11]. The operation uses four bytes as an input and is performed on all four bytes $a_{\{i,j,k,l\}}$ in parallel as shown in the hardware structure in Fig. 3.

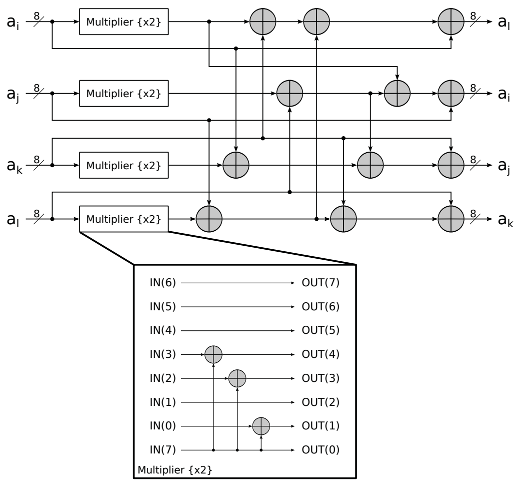

Fig. 3. Low-Area AES-MixColumns implementation.

With this structure, four bytes are handled per clock cycle. To perform this operation on all bytes within the accumulator, four clock cycles are needed. The most area-consuming part of this operation is the (de)-multiplexer structure to get the required bytes into the logic and to write them back to the right position in the register.

B. *AES S-Box*

The S-Box can be implemented as a ROM, but this approach is problematic when optimizing for low area and low power consumption. Another approach is to implement a combinatorial lookup table. This is rather easy to implement, but is inefficient in terms of area and has performance drawbacks because of long combinatorial paths.

Therefore, it is better to use an S-Box implementation based on optimized combinatorial logic such as the one described in [11]. This reduces the required area and decreases the critical path through the S-Box. Thus, area optimization also results in higher possible clock frequencies. The area of the LUT-based S-Box is 830 GE (S-Box is implemented with a one-stage pipeline to reduce the critical path), while the optimized S-Box only needs 550 GE (optimized combinatorial logic with one pipeline stage) when synthesizing for a target clock of 100 MHz.

With a single S-Box only one byte can be substituted per clock cycle. To parallelize the substitution process for all bytes of the accumulator, it is necessary to have as many S-Boxes as bytes to be substituted. Because this implementation focuses on yielding low area, only one S-Box is placed on the chip. To substitute all bytes of the accumulator, the substitution operation is serialized. This increases the operation's execution time to 17 clock cycles for substituting all 16 bytes.

IV. HOST CONTROL

To perform encryption or decryption, multiple steps have to be executed. For this reason, the hardware design uses a state machine for processing the data as shown in Fig. 4. Depending on the current state, the host has to perform different actions for further processing.

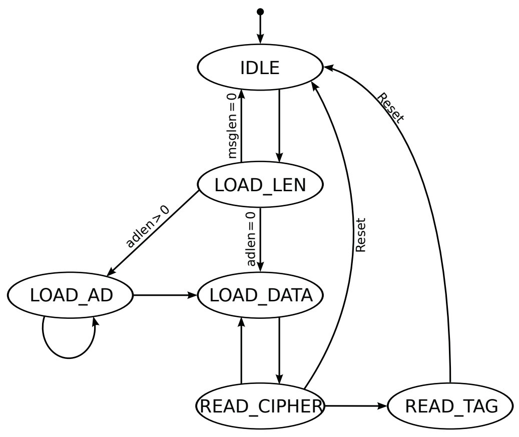

Fig. 4. State machine of the AEGIS128 chip.

The host communicates over an 8-bit AMBA APB interface with the chip. Since the algorithm-related registers are larger than 8 bit, they are concatenated from 8-bit bus transfers. Tab. I shows the registers available to the host and their respective address ranges.

TABLE I. REGISTER MAP OF THE AMBA-APB INTERFACE

Register	Address	Type
CONTROL	0x00	R/W
DATA / KEY	0x10-01F	R/W
TAG / IV / LENGTH	0x20-0x2F	R/W

The register in the address range 0x10 to 0x1F serves a different purpose depending on the current state. In state `IDLE`, it is used to write the key. In state `LOAD_AD`, it is used to write the associated data. Moreover, the register is used to load message blocks and read the en-/decrypted message blocks in the states `LOAD_DATA` and `READ_CIPHER`, respectively.

Similarly, the register in the address range 0x20 to 0x2F is specific to the chip's current state. When being in the `IDLE` state, this address range is used to write the initial vector. If the chip is in state `LOAD_LEN`, this address range is used to write the 64-bit lengths of associated data and message data as shown in Tab. II. When being in state `READ_TAG`, the computed message tag can be read from this register. Reading from this register in any other state returns 0x00 to avoid the leakage of any internal data processing. Writing to this register when not being in either the state `IDLE` or `LOAD_LEN` has no effect.

TABLE II. REGISTER MAP FOR SHARED LENGTH REGISTERS

Register	Address	Type
AD LENGTH	0x20-0x27	R/W
DATA LENGTH	0x28-0x2F	R/W

In addition to the algorithm-related registers, there is a `CONTROL` register. The host uses this register to read the current state and to control the chip. The layout of the `CONTROL` register is shown in Tab. III.

TABLE III. BIT LAYOUT OF CONTROL REGISTER

Bit	7	6	5	4
Type	-	R	R	R
Name	RFU	STATE[2]	STATE[1]	STATE[0]
Bit	3	2	1	0
Type	W	R	W	R/W
Name	Reset	Busy	Start	OP_Mode

State transitions as shown in Fig. 4 only happen if the user either writes the *reset* or the *start* bit in the `CONTROL` register. To maintain readability, state transitions from any state to `IDLE` by setting the *reset* bit are omitted. Most of the state transitions in Fig. 4 happen if the *start* bit is written. During state transitions, the host signals this with a `BUSY` state.

V. LOW AREA OPTIMIZATION

This work's implementation focuses on yielding lowest possible area. Several measures are taken into account to decrease the number of required gates. One such measure is register sharing. Since the key K_{128} and the initial vector IV_{128} are only needed once during initialization, they are loaded into the `DATA` and `TAG` registers, respectively. The `TAG` register is furthermore used for storing the lengths of the associated data (*adlen*) and the message (*msglen*). Only when being in state `READ_TAG`, the message tag is stored in this register. With this technique 384 bit of registers can be saved.

Another measure to reduce area is the appropriate alignment of registers that are larger than 8 bit. Since the AMBA APB interface has 8-bit width, write-enable signals need to be generated to write an incoming byte to the right position in the register. For the 128-bit input/output registers of the AEGIS128 chip, the lower bits of the address can directly generate the write-enable signal if the registers are aligned to addresses that are a multiple of 16.

Combining these techniques with optimized implementations of `AES-SubBytes` and `AES-MixColumns` decreases the required area by 20 %.

VI. FPGA IMPLEMENTATION

In addition to the ASIC implementation described before, we provide a fully running design for a Xilinx Artix-7 FPGA. This design reuses the ASIC design. For communication, an UART (Universal Asynchronous Receiver Transmitter) interface is used. This requires an additional layer to translate the UART data to AMBA APB commands. The structure of the respective design is shown in Fig. 5.

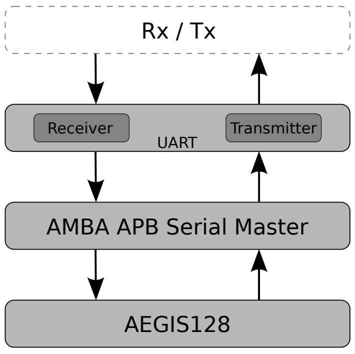

Fig. 5. Top-level structure of FPGA implementation of AEGIS128.

To write a byte to a specific address of the AMBA APB bus, the host needs to write three bytes over UART: first, a control byte indicating a write transfer, second, the address, and third, the data byte to be written. Reading a byte from a specific address is done accordingly. The host first needs to write two bytes over UART: the control byte indicating a read transfer and the address. Then, the host can read the data byte from the APB interface.

The proposed design, as shown in Fig. 5, is successfully synthesized with a clock frequency of 100 MHz. It uses 727 slices on a Xilinx XC7A100T-1CSG324-3 FPGA. If the clock frequency is increased to 130 MHz, the design requires 859 slices.

VII. RESULTS

The implementation uses a 0.13 µm low-leakage CMOS process from Faraday Technology [3]. The final layout is shown in Fig. 6.

Synthesis and Place and Route (P&R) was done using Cadence design tools. With the optimizations described in Section 3, the minimum area of the implementation after synthesis is 13,558 GE. The design is then capable of clock frequencies of up to 100 MHz. After P&R the area increases to 14,481 GE. The difference in area is due to the added filler cells, the clock tree and other cells needed for P&R. For the specified process technology, this results in a chip size of 0.06942 mm^2 after synthesis and 0.07414 mm^2 after P&R. The proposed design has a throughput of 65 Mbps when

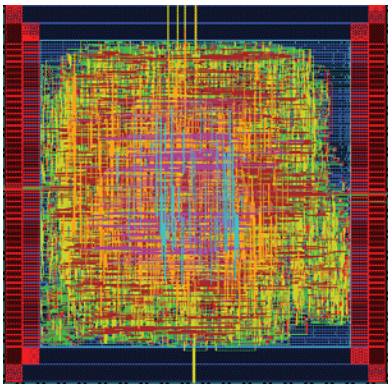

Fig. 6. Layout of AEGIS128.

operating at a clock frequency of 100 MHz. The maximum clock frequency was determined with 111 MHz. In this case, the chip area and throughput increase to 15,842 GE and 71 Mbps, respectively.

As shown in Tab. IV, 189 clock cycles are needed to process one block of 128-bit associated data. To encrypt or decrypt one 128-bit message block, 197 clock cycles are required. Moreover, Tab. IV presents the runtime information for initialization and finalization. The cycle counts already include the needed cycles to write to or read from the AMBA APB interface.

TABLE IV. CLOCK CYCLES FOR DIFFERENT OPERATIONS.

Operation	Cycles
Initialization	1,374
Associated Data	189
Encrypt/Decrypt	197
Finalization	863

A detailed area estimation of all components is shown in Tab. V. The datapath, which contains all registers, is the most area-consuming part.

TABLE V. AREA ESTIMATION AFTER SYNTHESIS.

Component	GE
Interface	498
Algorithm	9,950
Algorithm : Control Unit	1,755
Algorithm : Datapath	11,305
Algorithm : Datapath : ALU	3,314
Sum	13,558

Since the algorithm is based on a 640-bit state and data processing is done using 128-bit blocks, the resulting chip is larger than comparable AES implementations such as described in [4]. An implementation of AES-CCM optimized for low energy is shown in [5]. Yielding a maximum throughput of 54 Mbps, their design consumes 14,936 GE. Consequently, authenticated encryption using AES-CCM consumes more chip area than using AEGIS128. Moreover, the AES-CCM implementation in [5] does not provide authenticity for associated data. With a clock frequency and throughput of 111 MHz and 72 Mbps, respectively, the hardware implementation of AEGIS128 outperforms AES-CCM by roughly 33 %.

VIII. CONCLUSION

As part of the CAESAR competition, AEGIS128 is one promising candidate for a new authenticated encryption algorithm. The paper describes the first ASIC implementation of AEGIS128. In this context, we proposed techniques to reduce the required chip area. In particular, register sharing, optimizations of the basic AES routines, and a combinatorial S-Box are incorporated to minimize area. Application of these techniques decreased the required area by 20 %, ultimately resulting in a chip area of 13,558 GE.

ACKNOWLEDGMENT

The work has been supported in part by the Austrian Science Fund (project P26494-N15).

REFERENCES

[1] ARM Limited. AMBA 3 APB Protocol Specification, 2001.

[2] D. Bernstein. Cryptographic Competitions. http://competitions.cr.yp.to/, 2014.

[3] Faraday Technology Corporation. FSC0L_D 0.13 µm Standard Cell Library, 2006.

[4] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES implementation on a grain of sand. *Information Security, IEE Proceedings*, 152(1):13–20, Oct 2005.

[5] L. Huai, X. Zou, Z. Liu, and Y. Han. An Energy-Efficient AES-CCM Implementation for IEEE802.15.4 Wireless Sensor Networks. In *Networks Security, Wireless Communications and Trusted Computing, 2009. NSWCTC '09. International Conference on*, volume 2, pages 394–397, April 2009.

[6] National Institute of Standards and Technology. Advanced Encryption Standard, FIPS 197, November 2001.

[7] National Institute of Standards and Technology. Recommendations for Block Cipher Modes of Operation: Methods and Techniques, NIST Special Publication 800-38A, December 2001.

[8] National Institute of Standards and Technology. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, NIST Special Publication 800-38B, May 2005.

[9] National Institute of Standards and Technology. Recommendations for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D, November 2007.

[10] F. N. Whiting D., Housley R. Counter with CBC-MAC (CCM), IETF RFC 3610, September 2003.

[11] J. Wolkerstorfer, E. Oswald, and M. Lamberger. An ASIC Implementation of the AES SBoxes. In B. Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 67–78. Springer Berlin Heidelberg, 2002.

[12] H. Wu and B. Preneel. AEGIS: A Fast Authenticated Encryption Algorithm. In T. Lange, K. Lauter, and P. Lisoněk, editors, *Selected Areas in Cryptography – SAC 2013*, Lecture Notes in Computer Science, pages 185–201. Springer Berlin Heidelberg, 2014.