Authenticated Encryption – A Hardware Designer's Perspective

Elif Bilge Kavun[1], Hristina Mihajloska[2] and Tolga Yalçın[3]

[1] University of Sheffield, Sheffield, United Kingdom
[2] Ss Cyril and Methodius University, Skopje, North Macedonia
[3] Northern Arizona University, Flagstaff, AZ, USA

Abstract. Authenticated encryption (AE) has been a vital operation in cryptography due to its ability to provide confidentiality, integrity, and authenticity at the same time. Its use has soared in parallel with widespread use of the Internet and has led to several new schemes. There have been studies investigating software performance of various schemes. However, the same is yet to be done for hardware. We present a comprehensive survey of hardware (specifically ASIC) performance of the most commonly used AE schemes in the literature. These schemes include encrypt-then-MAC combination, block cipher based AE modes, and the recently-introduced permutation-based AE scheme. For completeness, we implemented each scheme with various standardized block ciphers and/or hash algorithms, and their lightweight versions. Our evaluation targets minimizing the time-area product while maximizing the throughput on an ASIC platform. We used 45 nm NANGATE Open Cell Library for syntheses. We present area, speed, time-area product, throughput, and power figures for both standard and lightweight versions of each scheme. We also provide an unbiased discussion on the impact of the structure and complexity of each scheme on hardware implementation. Our results reveal 13-30% performance boost in permutation-based AE compared to conventional schemes and they can be used as a benchmark in the ongoing AE competition CAESAR.
Keywords: authenticated encryption, hardware performance, authenticated encryption mode, CBC-HMAC algorithm, permutation-based authenticated encryption

1 Introduction

Encryption without authentication is *NOT* secure.

Internet Protocol Security (IPSec) protocol suite [23] offers encryption without authentication as a recommended and supported option, which has tempted the researchers to extensively study security of encryption-only schemes over the years. Each study resulted in the inevitable conclusion above: Authentication is crucial for a completely secure communication channel [6,10]. In the absence of authentication, attackers can forge messages by cut-and-pasting different parts of encrypted messages.

In today's world, secure communication is crucial. The online banking transactions must be securely carried on. Information exchanged via email communication has to stay confidential. We need to be able to fully trust the communication channel with two main issues in mind: Security – to protect the contents of the message from an adversary – and authentication – to ensure that the message is genuine.

The simplest solution is to use encrypt-then-MAC paradigm [24], which is considered as the *right* way to compose a secret key encryption and a MAC into an authenticated encryption [3]. Over the past two decades, several strong ciphers and hash functions have been proposed [1,7,11, 12,15,31,41,43] and even standardized. They can be used together in an encrypt-then-authenticate scheme. While this provides guaranteed security, the same cannot be said for its efficiency and performance. Using two completely different algorithms for encryption and authentication requires

This work was done while the authors were working (H. Mihajloska as visiting researcher) at Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum, Germany. The research was supported in part by the DFG Research Training Group GRK 1817/1.

This is the IACR ePrint version of *Elif Bilge Kavun, Hristina Mihajloska, and Tolga Yalçın. A Survey on Authenticated Encryption—ASIC Designer's Perspective. ACM Comput. Surv. 50, 6, Article 88, December 2017.* Please refer to the original paper for citations (`https://doi.org/10.1145/3131276`).

implementation of both algorithms separately. This not only means additional implementation effort for each algorithm, but it also means additional code space or silicon area in a software or hardware implementation, respectively.

In order to overcome this obstacle, combined schemes have also been proposed. Most of these schemes use a block cipher for encryption with additional invocations for authentication [5, 16, 34, 39]. Some of these existing schemes have been well-analyzed and recommended by NIST [16, 34]. Some have been standardized or referenced within another standard [39]. More recently, using permutation-based sponge functions for authenticated encryption has been proposed [8] and attracted a lot of attention due to the SHA-3 competition being won by a sponge hash function [42], namely Keccak [7]. There are also ciphers designed specifically for authenticated encryption [13, 18]. However, a unified scheme or family of schemes is yet to be proposed. This open question and increasing interest eventually led to the ongoing authenticated encryption competition CAESAR [14].

The design of an authenticated encryption scheme is a trade-off between many different aspects including security level, execution speed, area cost, and power. Software performance of authenticated encryption schemes is a fairly well-studied area. One study among the most recent works is the FSE 2011 paper by Krevetz and Rogaway [29], which presents a software performance comparison of well-known authenticated encryption modes to the OCB3 mode [30]. Another software evaluation paper published by Gouvêa and López at LatinCrypt 2012 [19] compares the performances of many authenticated encryption schemes on MSP430X family of microcontrollers providing some interesting results. In other works [5, 8, 13, 16, 18, 34, 39], authors present both hardware and software performances of their own proposals. On the hardware side, the situation is much worse. Until now, there have been evaluation studies of underlying cipher and hash functions for very specific design targets such as lightweight and low-latency encryption, etc [17, 28]. A very recent study has presented results on the permutation-based authenticated encryption for lightweight applications [46]. To the best of our knowledge, a comprehensive hardware evaluation of different types of authenticated encryption schemes does not exist in literature. In this study, we aim to bridge this gap by providing a comprehensive survey on the performances of existing *well-known, widely-used and secure* authenticated encryption schemes from a hardware, specifically Application-specific Integrated Circuit (ASIC), designer's perspective. Our observations include the results for speed, area, time-area product, throughput, and power. We also propose unbiased recommendations for hardware-efficient authenticated encryption schemes, in the light of our results.

The rest of the paper is organized as follows. In Section 2, we provide short descriptions of all investigated authenticated encryption schemes. Note that we do not evaluate existing authenticated encryption cipher proposals such as [9, 13, 18] in this work and only focus on alternative schemes. Hardware implementation specific details of all schemes are given in the following hardware evaluation section. Section 4 presents and discusses the implementation results with different metrics. Finally, we conclude our paper in Section 5 with recommendations on hardware-efficient authenticated encryption scheme design.

2 Authenticated Encryption Schemes

We investigate the Authenticated Encryption (AE) schemes in three groups: The first group covers the most commonly used and straightforward encrypt-then-MAC scheme, which is realized by using a block cipher and a hash function. In this group, we look only at the CBC-HMAC scheme [33] mainly due to its widespread use in IPSec protocol suite. The second group consists of the block cipher based authenticated encryption schemes. Out of several existing schemes in the literature, we have picked up the most commonly used ones – namely CCM, GCM, EAX and OCB [5, 16, 30, 34, 39], some of which are even standardized or recommended by NIST. The

last group is the permutation-based authenticated encryption schemes [8], which were proposed with the introduction of sponge functions. It is a very new research area, but these schemes exhibit some very advantageous properties in terms of hardware implementation. In the following subsections, each of these schemes is described briefly.

2.1 Encrypt-then-MAC

As the name implies, this scheme relies on first encryption and then authentication of data as given in Bellare and Namprempre's paper [4]. The encryption-authentication sequence can be applied either per data block (or a few blocks) or on the whole data package, depending on the application requirements. It can be realized with several different underlying schemes. For example, any of the counter, offset-feedback or cipher-block-chaining modes can be used for encryption. The same is also valid for authentication, resulting in various combinations. In our case, we will focus only on the CBC-HMAC [33] scheme. It is perhaps the most commonly used AE scheme within the IPSec protocol suite. Modern implementations rely on AES as the block cipher and SHA-1 as the hash function, while in the recent years SHA-1 has been mostly phased out and replaced with SHA-256. However, this scheme is not limited to 128-bit block size ciphers; so, its realization via modern lightweight ciphers is also within our area of interest. In our case, we have chosen AES-CBC/HMAC-SHA as the main combination and it is accompanied by the lightweight version composed of the PRESENT-CBC/HMAC-Keccak combination. Descriptions of ciphers and hash functions in these combinations are given as follows:

- **Standard Approach:** The standard approach is realized with the AES-CBC/HMAC-SHA, where the 128-bit key version of AES is used together with SHA-256. Since the two modules have completely different number of cycles and block sizes, their operation requires careful scheduling. AES input block is 128 bits, while for SHA-256, it is 512 bits. This means four AES operations can be performed during one SHA operation in order to keep the two modules synchronized. Each AES output has to be stored in a buffer, which means additional storage. Furthermore, the AES module has to stay idle after every 40 cycles of AES operations until 64 cycles of parallel running SHA-256 operation are completed. The alternative is to run them literally in an encrypt-then-MAC order. This would increase the total number of cycles from 64 (maximum of the two modules) to 104 (sum), but it will also remove the need for additional storage, by making use of the SHA-256 message digest register as the AES result buffer. We have tried both options, and opted for the latter one due its better time-area product.
- **Lightweight Approach:** In the lightweight approach, we chose the PRESENT cipher with 80-bit key option for encryption, and 200-bit version of the SHA-3 winner Keccak for authentication. This choice comes with the same data block size for both modules. Additionally, the total number of rounds for PRESENT-80 and Keccak-200 are 31 and 18, respectively, which makes the synchronization very simple. In this implementation, each encrypted message block is sent to authentication, which finishes in 18 cycles and stays idle for the next 13 cycles while the next message block is being encrypted. This configuration offers a very compact implementation with ample and proven security using standardized algorithms.

2.2 Block Cipher Based Mode of Operations

In almost all modern applications, authenticated encryption is obtained by the use of a block cipher to realize both encryption and authentication. This scheme is very advantageous in terms of both software and hardware implementation. Instead of implementing and using resources for two separate algorithms, only a single block cipher is implemented on the target platform together with the accompanying mode wrapper and the extra storage required for intermediate encryption and authentication states. Usually, only a single key material is required for the chosen

block cipher. Depending on the mode, either the cipher function is run once per each encryption and authentication, or both of them can be incorporated into the same cipher run. Most of the existing implementations use the NIST-standard AES [1] as the underlying cipher function. Some modes are even specified to be used only with AES. But most of them can also be used with another 128-bit block size cipher, while some of them are specified to be used with 64-bit block size ciphers as well. In our study, we have chosen CCM [16], GCM [34], OCB3 [30], and EAX [5] for investigation due to their widespread use in commercial applications.

Although most of these applications realize these modes with AES, we have decided to implement them using Clefia with 128-bit key (as the second 128-bit block size cipher), PRESENT with 80-bit key and mCrypton with 96-bit key (as 64-bit block size ciphers), as well. Clefia and PRESENT are chosen since they are the *ISO-standardized* [25] lightweight ciphers. On the other hand, mCrypton [31], although not an official standard, is a very widely-used cipher in industry. Before going into the details of the target modes, we briefly introduce these ciphers.

- **AES** is the well-known and popular NIST-standard 128-bit block cipher. It comes with key lengths between 128 and 256 bits with increments of 32 bits. The most commonly used and benchmarked version is 128-bit key version that has 10 rounds and we shall also be using it in our study.
- **Clefia** is the 128-bit block size cipher developed by Sony Corporation. It has a generalized Feistel structure with four data lines. Supports different key lengths: 128, 192 and 256 bits. In our work, we implement and use the 128-bit key version of Clefia with 18 rounds.
- **PRESENT** is the ISO-standard 64-bit lightweight block cipher that comes with two different key lengths: 80 and 128 bits. In this work, we only use the 80-bit key version with 31 rounds.
- **mCrypton** is a 64-bit lightweight block size cipher with 12 rounds and three different key lengths: 64, 96, and 128 bits. While the 64-bit version is deemed sufficient for most lightweight applications, we have opted to use the 96-bit key version in order to provide a security level not lower than that of the PRESENT core.

Next, we will provide brief descriptions of the block cipher based authenticated encryption modes used in our evaluation.

- **CCM** is the abbreviation for **C**ounter with **C**BC-**M**AC. CCM authenticated encryption with associated data essentially combines the counter (CTR) mode of encryption [32] with CBC-MAC authentication scheme [26]. To process each message block, a counter is encrypted with the underlying block cipher and the result is XORed to the message for ciphertext production. Its final state after all blocks are processed is the authentication tag. At the end of processing of each message block, the counter is also incremented for the next message block encryption. As a result of this operation sequence, each message block requires two encryption runs, resulting in low throughput. It can be improved by using two encryption module instances, which doubles the resource use. This is a primary trade-off in CCM implementations. On the positive side, decryption can also be realized using the same encryption scheme with ciphertext data instead of plaintext. Therefore, only the encryption functionality of the underlying cipher module is sufficient for CCM operation.
- **GCM** is the abbreviation for **G**alois **C**ounter **M**ode. It is very similar to CCM in operation. The encryption stage is identical, but authentication is realized via multiplication in GF(2^{128}) instead of the second encryption in CCM. Only a single encryption is required per message block, which results in a high throughput. The only penalty is the additional finite field multiplication, which can be effectively implemented by distributing the steps of multiplication to several cycles in parallel with the encryption operation. GCM mode is being deployed more and more in many applications due to its obvious advantages over CCM.

– **OCB3** is the abbreviation for a modified version of **O**ffset **C**ode**b**ook mode. OCB3 also employs GF(2^{128}) as in the case of GCM, but in a simpler way. It does not require full multiplication, but only multiplication by powers of z (the variable used in the polynomial representation of the finite field elements). In order to process the message block i, OCB3 performs a finite field multiplication of a nonce/key-dependent constant L_0 by the polynomial z^j, where j is the number of trailing zeros in the binary representation of the block index i. The result is known as tweak. Since the tweaks are computed prior to OCB3 operation and stored in the memory, the whole OCB3 operation is executed very effectively. OCB is usually referred to as the highest throughput mode. However, it has drawbacks as well. Unlike CCM and GCM, it requires both encryption and decryption functionality in the underlying cipher. Furthermore, it is not a license-free scheme and can only be used subject to royalty fees except certain applications.
– **EAX** authenticated encryption mode is a two-pass scheme: Encryption and authentication are performed separately. This makes EAX mode much slower than GCM or OCB3 modes. On the other hand, EAX uses only the *encrypt* functionality of the block cipher, which makes it easy to fit into constrained implementations.

In our study, we implemented both standard and lightweight constructions of the target modes using the above mentioned block ciphers. By *standard*, we refer to the 128-bit key size block cipher versions (128-bit block size), while *lightweight* corresponds to the 80/96-bit key size cipher versions (64-bit block size). Since CCM and OCB3 modes are strictly specified for 128-bit block size ciphers, we implemented them using AES and Clefia and skipped the 64-bit block size ciphers for these two modes. For GCM and EAX, we used all four target ciphers.

When we have AES as the underlying cipher, encryption and authentication of each message block requires two cipher module calls for CCM and EAX, and a single call for GCM and OCB3. In case of OCB3, decryption of the encrypted message block requires decryption functionality of the AES core. Therefore it requires execution of key expansion prior to each decryption session in order to generate the decryption key schedule. This also means an AES core which requires more resources in both software and hardware implementation.

In constructions where Clefia is used, a key initialization has to be executed prior to each session regardless of encryption or decryption operation. On the positive side, a separate decryption key schedule is not required. It can be generated on-the-fly from the encryption. However, the higher resource requirement of the Clefia core for OCB3 (due to both encryption and decryption functionalities) is still valid.

In the case of PRESENT, the finite field multiplication in GCM can be split into a long time due to the high number of rounds in PRESENT, making the whole scheme even more resource-effective. The only drawback is the overall low throughput, which is a natural result of the high number of rounds. However, for most lightweight applications, this does not pose a serious problem.

Despite having an identical construction, mCrypton results in higher throughput compared to PRESENT due to much lower number of rounds required by mCrypton. However, it should also be noted that mCrypton implementation requires more resources than PRESENT.

Most of the hardware implementations in literature are based on the AES as underlying primitive. Numerous implementations of AES wrapped with various authentication modes exist on different platforms. As we are interested only in hardware evaluation (on ASIC) in this study, we investigated the available ASIC implementations. For the CCM mode, a few implementations have been reported both in academia and industry. One commercially available compact core for AES-CCM by Helion [21] consumes 19 kGE on ASIC ($0.13\mu m$ CMOS) platform. The implementation given in [37] is running at 264 MHz and achieves a throughput of 2.69 Gbps for 20.5 kGE. The most compact known implementation of AES-CCM is presented in [13], where the authors use

the lightweight AES engine in [35] and report 3.4 kGE for area consumption at a clock frequency of 20 MHz on STMicroelectronics 65 nm CMOS technology.

For AES-GCM, there exist three commercial implementations with different levels of performance on ASIC by Helion [22]. They occupy 13 kGE, 19 kGE, and 39 kGE depending on the number of clock cycles that are used to encrypt/decrypt one block (128 bits) of data. On the other side, in [40], the authors propose a compact implementation of AES-GCM that consumes 34.5 kGE of area while achieving a throughput of 2.56 Gbps on $0.13\mu m$ CMOS technology. The other reported implementations in the literature are based on high-speed evaluations with a throughput of more then 10 Gbps.

Most of the available implementations for AES-OCB are done for FPGA. The authors of the paper [13] implemented a lightweight ASIC version of AES-OCB2 that requires 5.9 kGE with 226 clock cycles per one block of message. Furthermore, [38] provides hardware (ASIC) performance results for AES-EAX.

2.3 Permutation-based Authenticated Encryption

An n-bit block size cipher is practically an $n+K$-bit permutation (K being the key size) with no diffusion from data state to key state. This property together with zero need to limit diffusion allows a block cipher to be used in iterated permutation mode. In such a construction, the key schedule is removed and the block cipher is replaced by an $n+K$-bit permutation, resulting in a block cipher without inverse. Such a construction has the capability to perform both encryption and message authentication, or when combined, authenticated encryption. Recently proposed sponge functions present very resource-efficient building blocks for this type of authenticated encryption scheme.

The first proposed permutation-based authenticated encryption scheme was based on Keccak with SpongeWrap [8] configuration. However, it was a generic construction that can be used with any sponge function. Therefore, in our study, we have investigated the possibility of using existing sponge functions Keccak, Photon [20], Quark [2], and Spongent [11]. In the following, we briefly introduce these sponge-based functions.

- **Keccak** is a family of sponge functions that has been standardized by NIST. The state can be 25, 50, 100, 200, 400, 800, or 1600 bits. Suitable for lightweight purposes is Keccak-f[200]-64, where the output size is 64 bits. The total number of rounds of this version is 18, and gives security strength of 80 bits.
- **Photon** is a hardware-oriented hash function family. The hash output sizes are: 80, 128, 160, 224, and 256 bits. The internal state size depends on the hash output size and can take only 5 distinct values: 100, 144, 196, 256, and 288 bits. In our work, we decided to work with Photon-160/36/36 because it provides 80-bit collision resistance. The number of rounds of this version is 12.
- **Quark** is a lightweight hash function family. It comes with three different versions: u-Quark, d-Quark and s-Quark. For each, its rate, capacity, digest length are different; but its internal functions are the same. d-Quark is the one with 80-bit security against all attacks, and we take it into our evaluation. It has parameters 16-bit rate, 160-bit capacity, and 176-bit message digest. The number of rounds needed to be implemented in this version is 704.
- **Spongent** is another lightweight hash function family. The authors proposed 13 Spongent variants -– for different levels of collision and (second) preimage resistance as well as for various implementation constraints. To be consistent, SPONGENT-88/176/88 is used in our evaluation where the rate is 88 bits, capacity is 176 bits, digest is 88 bits, and the number of rounds is 135.

It is a difficult task to provide the same security level in all these sponge-based hash functions. Therefore, we had to leave the standard implementations out. Instead, we implemented only their lightweight versions. Minimum 80-bit security is provided in all implementations. For each function, the rate is selected to be a multiple of 16 bits; i.e., 32 bits in Keccak-200 and Photon-196, and 16 bits in Quark-176 and Spongent-176.

All permutation-based AE schemes have the same working principle. The SpongeWrap construction is virtually identical to standard sponge hash operation in encryption mode. At startup, the key divided into 32-bit blocks and each of the blocks are added to the internal state (starting with zero initial value) and permuted. It is followed by the addition of optional associated data (AD) and message blocks. After addition of each 32-bit block, the permutation is executed for the specified number of rounds (e.g., 18 for Keccak). This step is known as the absorption. During the absorption of message blocks, previous value of the state is added to the current message (plaintext) block to generate the corresponding ciphertext block. In the case of decryption, instead of adding the message on to the internal state, ciphertext is sent directly to the permutation module. 32-bit tag is extracted from the final internal state. In case longer tags are needed, the permutation function is run several times and 32-bit chunks of tag are extracted following each run. This process is known as squeezing.

3 Hardware Evaluation

In our hardware evaluation process, we mainly targeted minimizing the area and maximizing the throughput for each scheme. As a result, we opted for a round-based implementation approach. This way, we achieved the smallest and fastest implementations for block ciphers and hash functions. However, for completeness, we also investigated options for unrolled implementations where more than one round of a cipher is executed within a single cycle. For example, we tried 2 and 4 rounds within a single cycle for each of the block ciphers under investigation. As expected, these implementations resulted in larger areas, but also higher throughput, and hence better time-area product compared to a round-based implementation. On the other hand, implementation of more than a single round within a cycle caused longer combinational delay paths and therefore lower maximum operating frequency. As a consequence, while per cycle based throughput was higher than a single round design, in all other options, due to the lower maximum operating frequency, effective throughput dropped dramatically resulting in a worse time-area product performance.

As an example, round based implementation of the AES cipher resulted in an area of 16.1 kGEs, a maximum operating frequency of 268.6 MHz, encryption in 10 cycles and a time-area product of 602 ns $\cdot$ kGE per encryption. On the other hand, a 2-round per cycle unrolled version of AES resulted in an area of 25.6 kGE, a maximum operating frequency of 145 MHz, encryption in 5 cycles and a time-area product of 884 ns $\cdot$ kGE per encryption. Clearly, round-based design gives better time-area product performance. The figures get worse as the number of rounds per cycles increases (for example 1766 ns $\cdot$ kGE per encryption for a 5-round per cycle version). In summary, by targeting a low area and high throughput, we have reached the best (lowest) time-area product as well.

Before our evaluation, we checked several comparison studies and observed that most authors prefer to implement their design(s) in one (or very few) specific configuration and using whatever technology library they have. They then compare their implementation(s) with other implementations that might have been implemented using different approaches and/or technology libraries. We decided that such a comparison without common design approach and implementation target(s) would not be fair at all. Such an approach usually results in an *apples and oranges* type of comparison and ends up favoring the author's own design. A very simple example would be freezing the flip-flop states during certain operations: In industrial designs, this is usually managed by using multiplexed flip-flops (i.e. scan flip-flops), which in turn increases the effective flip-flop

area. However, some authors use clock gating together with regular flip-flops in order to come up with smaller chip areas. Such an approach would probably not be applicable in a commercial design, where the circuit must pass all the verification tools in the design flow, which are not always compatible with gated-clocks [27, 44].

Instead, we decided to have a common design approach and structure (at least where possible) for all the algorithms we were to evaluate. In order to achieve this, we implemented a separate optimized wrapper for each of the *Encrypt-then-MAC*, *Block Cipher Based Mode of Operation*, and *Permutation-based Encryption* categories. Even in this case, due to slight variations in the algorithm definitions, it was not possible to design a universal wrapper for each category. In each case, we tried our best to keep a unified interface. Moreover, we used the same state-freezing scheme (a single positive-edge triggered clock for all parts of the circuit), the same resetting scheme (active-low asynchronous reset for all flip-flops), the same clock-gating for all flip-flops (via multiplexing inputs), the same control logic approach (start-ready operation, where the circuit becomes active with a start pulse, goes through various phases, i.e. initialization, hashing, finalization, etc., and generates a ready pulse and halts upon completion of the requested operation), and so on. However, even so, it was not always possible or cost-effective to apply the same structure on all the algorithms that fall in the same category. In such cases, we had to implement slight algorithm-specific optimizations to obtain the best possible time-area trade-off. We, therefore, believe that we managed to come up with the fairest comparison – even though not always with state-of-the-art results.

Furthermore, we decided to use a publicly available standard-cell library to provide a common reference point accessible by everyone. Therefore, we opted for 45 nm generic NANGATE [36] Open Cell Library. We performed hierarchical synthesis for our designs using both Cadence Encounter RTL Compiler v10.1 and Synopsys Design Compiler vE-2010.12-SP2. The figures from both synthesis tools are close within a $\pm 7\%$ margin (in favor of either tool with no specific pattern). Since neither tool has an apparent advantage over the other, we only report the synthesis figures from Cadence. In all syntheses, typical operating conditions were assumed and both minimum area and maximum speed were specified as design targets. Note that all schemes are coded in Verilog HDL and tested with the test vectors given in the associated standards and/or specifications. In the absence of test vectors (for example, as in the case of PRESENT-CBC/HMAC-Keccak), we generated them using Matlab in a way to mimic the behavior of the corresponding scheme.

Each scheme was first synthesized for minimum area and maximum speed. Synthesis results, which are taken from the generated synthesis reports, are presented in Table 1 in Section 4. The area numbers in the table are given in terms of two-input NAND gate equivalents (GEs). Following the synthesis, generated netlists were used to simulate the actual module with 100 random keys together with 10 random plaintexts per key to get the best statistics. From these simulations, SAIF files were generated, which also contain the toggle counts. Both functional and post-synthesis physical simulations were performed using Modelsim v6.6c. In the last step, the SAIF files were sent back to the synthesis tool together with the netlist from the initial synthesis to run power analysis. The results of this power analysis are also presented in Section 4.

Note that we did not actually perform any place and route.

3.1 Design Choices

It would be prudent to briefly discuss design choices before going into details for each scheme. Firstly, we wanted to have a comparison of standard and lightweight schemes. When it comes to terms, there may be confusion regarding the design and/or algorithm parameters. For example, both PRESENT and Clefia have been accepted as lightweight block cipher standards by ISO. However, PRESENT block size is 64 bits, while it is 128 bits for Clefia. This begs the next logical question: What would be the block size for a cipher algorithm to be considered as standard or

lightweight? A similar question can also be asked for the key size: Which key size should we select for a standard or lightweight algorithm that offers more than one choice?

While our reasoning may be debatable, we had to make hard choices in terms of definitions and design parameters. After investigating several publications (including both scientific and commercial ones), most of which are among the references addressed in this work, and consulting with people from both academia and industry, we decided to focus on the algorithms at hand. A detailed discussion of algorithm choices is already presented in Section 2. In summary, our choice of schemes solely depend on user cases. The most common use of publicly available and standardized cryptographic schemes is in communication products, which rely on the IPSec Protocol for data security and privacy [23]. However, IPSec Protocol Suite fact is not a single standard, but a collection of request-for-comments (RFCs) [45]. Which of the standard algorithms are the most commonly used ones? This can, in theory, be determined by monitoring the Internet traffic on a certain hub. This not only is a completely different task itself, but it also differs from application to application and from site to site. For example, the Internet traffic of a residential site would have much lower IPSec utilization compared to that of a commercial or government site, resulting in false statistics. It would be ideal to monitor or obtain statistics from a big Internet service provider. Unfortunately, they do not give away these numbers freely. Instead, we had to rely on informal information obtained from various semiconductor vendors that produce and/or use security modules in their products.

As a result, we decided to focus on our existing schemes. As for the choice of design parameters, we followed a similar approach and decided to classify 128-bit block size ciphers as *standard* and 64-bit block size ciphers as *lightweight*. When it comes to the use of a hash function together with a cipher (as in encrypt-then-MAC scheme), the choice came naturally. In case of AES-CBC/HMAC-SHA, it takes 40 cycles of AES-128 operation to process 512-bits of data. While this data is being processed by the SHA-256 module in 64 cycles, AES module prepares the next 512-bit data. There is a minimal loss of clock cycles (64-44=20 cycles), which results in an almost seamless pipeline operation between the two modules that make up the scheme. The same also applies to PRESENT-CBC/HMAC-Keccak, where 31-cycle encryption can work in parallel with 18-cycle hashing by Keccak-200.

We were also careful in keeping the security levels the same for both encryption and hash modules, i.e. 128-bit AES and 256-bit SHA-256 (256/2=128-bit security) have the same security levels. Similarly, 80-bit (key size) PRESENT has similar security level as Keccak-200 with $r = 32$ ((200-32)/2=84-bits).

There are combinations we did not try, such as Clefia-CBC/HMAC-SHA or mCryption-CBC/HMAC-Keccak, or non-Keccak sponge-based hash functions for *Encrypt-then-MAC* scheme. Instead, we opted to present the readers with indicative numbers from major selections, which they can later interpolate to obtain the figures for other combinations as well.

3.2 Block Ciphers

We first started with the implementation of block ciphers. In this phase, we tried to come up with a unified input/output interface applicable to each cipher under consideration – with certain differences, such as message block size (128 vs. 64 bits) and key size (128, 96, or 80 bits). However, it turned out that this approach, while mostly successful, required more than just message block size change in certain cases. For example, Clefia required a pre-processing on the key (key expansion) and storing the pre-processed keys.

Apart from such exceptions, we were able to come with a generic interface, where the operation of each cipher starts with a start pulse, upon completion of operation, each module outputs its ciphertext together with a ready pulse. The next start pulse comes at the same time as the ready pulse resulting in zero-cycle loss, hence maximum throughput. The generic block diagram for the

block cipher cores is given in Figure 1 without the control module, whereas the detailed block diagram for the specific case of Clefia is given separately in Figure 2.

Note that, in all the block diagrams, we omit details such as bus width and detailed control signal labels, as these depend heavily on the block of interest. For example, for the parallel implementation of a 128-bit block size cipher like AES, the bus widths are set to be 128 bits, while for a 64-bit block size cipher, the bus width is reduced to 64 bits, too. We also use simplifications in labels, such as P, C, K to denote plaintext, ciphertext and key, respectively, as shown in the legend underneath Figure 1.

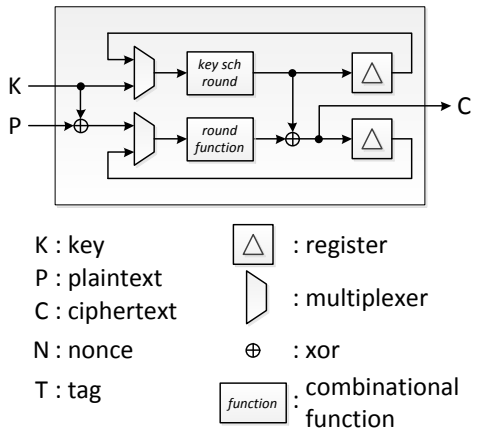

Fig. 1. Generic block diagram of the block cipher cores

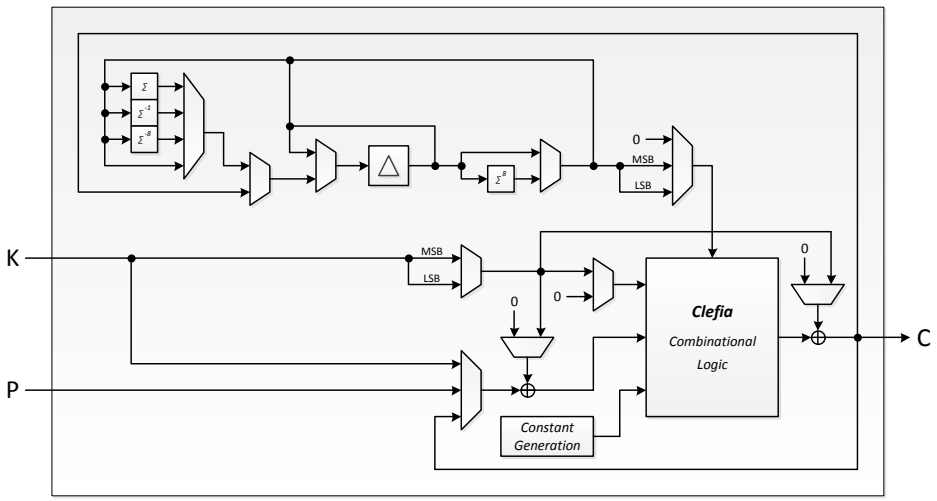

Fig. 2. Detailed block diagram of the Clefia block cipher core

3.3 Block Cipher Based AE Mode Wrappers

In the next phase (or upper layer), each block cipher is enclosed within a mode wrapper. The mode wrapper timings are identical to the underlying block ciphers. They too have zero-cycle loss. We also paid special attention in an attempt to come up with a minimalist wrapper design in order to keep additional area and delay caused by the wrapper at minimum. In this phase, we still assume that inputs to and outputs from the blocks are all parallel (i.e. 64 or 80 bits). In a practical application, the data to the security module is usually provided through a data bus controlled by a processor. Since data buses are usually limited to 32 or 64 bits (or even as low as 8 bits in embedded environments), there have to be additional registers for serial-to-parallel conversion (e.g. 32-to-128) at the inputs and parallel-to-serial conversion (e.g. 128-to-32) at the outputs of the wrappers. However, these are also very application-specific design parameters. Therefore, we did not implement them. It is left to the designers to estimate additional gate counts and add onto our numbers. In fact, this estimation can be as simple as multiplying the number of input/output bits with the area of a single multiplexed flip flop.

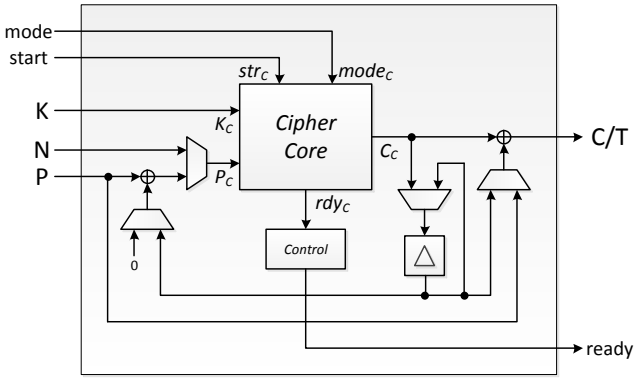

Fig. 3. Block diagram of the CCM authentication encryption scheme wrapper

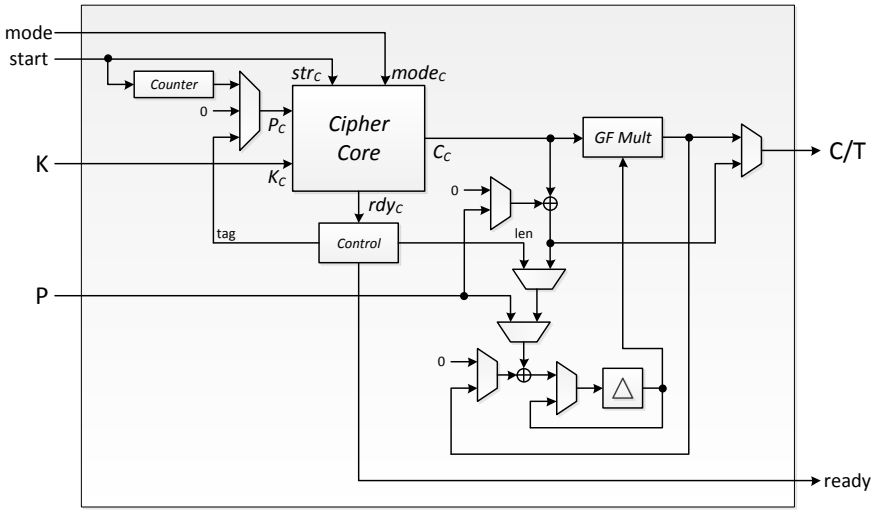

Fig. 4. Block diagram of the GCM authentication encryption scheme wrapper

As in the case of block ciphers, for the evaluation, we attempted to design a generic module on which any one of the four modes can be implemented by modifying the configuration of the multiplexers. Unfortunately, major differences between the modes cause significant additional overhead in order to utilize a single reconfigured wrapper. Therefore, we ended up using a separate wrapper for each mode, with various degrees of differences. The block diagrams for the wrappers are given in Figures 3 through 6, again without the control module.

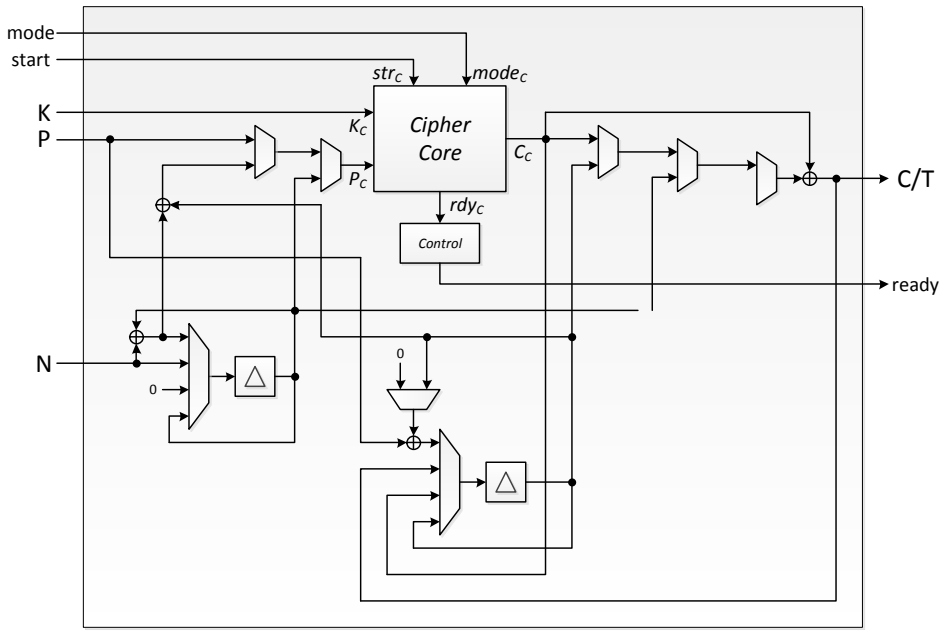

Fig. 5. Block diagram of the OCB3 authentication encryption scheme wrapper

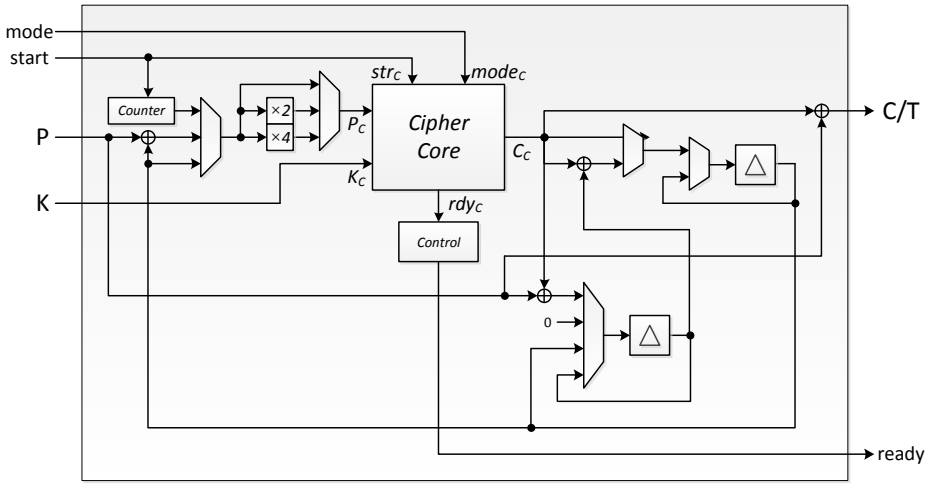

Fig. 6. Block diagram of the EAX authentication encryption scheme wrapper

3.4 Hash Functions

We followed an approach very similar to that of block cipher design in this phase. However, there had to be additional control signals to allow initialization of the hash function to its initial state. This part is copied from the wrapper design in the block cipher based AE mode wrappers. While the SHA-256 module required two levels of registers (one for the internal hash iteration, the other for the final hash variable), design of sponge based hash functions were almost identical to the block cipher design due to their underlying permutation structures.

3.5 Permutation-Based AE Modules

These modules were practically identical to the underlying sponge function, except for an additional multiplexer required to select between the plaintext and ciphertext in the encryption and decryption phases, respectively, as the input to the next permutation run. The block diagram of the permutation-based AE wrapper is given in Figure 7, also without the control module.

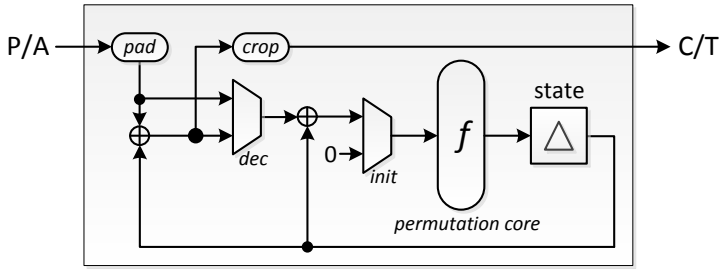

Fig. 7. Block diagram of the permutation-based authentication encryption scheme wrapper

3.6 Overall Strategy

Throughout the whole design, we used only flip-flops with positive-edge clocks and active-low reset. Each design is fully synchronous to a single clock input. We avoided any clock related design tricks targeted to keep the power consumption low (like gated-clocks), since in real designs such additions contradict with the overall testing strategy and are only employed at an upper level of hierarchy.

4 Results and Discussion

In total, we have evaluated 18 different schemes and combinations. In Table 1, we provide the area, speed, time-area product, throughput and power comparisons for all evaluated schemes. The results are also plotted in Figures 8 through 12 respectively for each metric. In our comparisons, we favor none of the schemes. The selection of the corresponding scheme is left to designers for the target application. Likewise, any researcher can use the results while designing a new authenticated encryption scheme either for a specific application or to come up with a general scheme. In the following, we present our unbiased comments on each design parameter.

– **Area.** As expected, lightweight permutation-based AE schemes result in the smallest area. Since the hash functions are practically part of the wrappers, no additional registers and only minimal extra combinational circuitry are required for wrapper implementation. This is in

contrary to most of the block cipher based schemes. Furthermore, the internal permutations functions have much simpler structures. On the other hand, the largest area belongs to GCM-AES AE scheme, which comes as no surprise because of the additional Galois field multiplier. However, it should also be noted that the schemes with the largest area results are also the schemes currently accepted as industry standards and hence are in wide use.

- **Speed.** Speed results are similar to area results, i.e., compact designs are also fast.
- **Time-Area Product.** Again, permutation-based AE schemes present the smallest results, except for the Quark-based one due to the high number of rounds in this hash function. Keccak-based schemes (Keccak AE and PRESENT-CBC/HMAC-Keccak) give the best results, which is also not surprising considering the low number of rounds and simple permutation function of Keccak. On the block cipher based AE side, OCB3 gives the best results as previously claimed in literature. However, since OCB3 is defined for 128-bit block ciphers, it cannot be used with lightweight ciphers for which GCM provides good results.
- **Throughput.** The results here are similar to time-area product results. Keccak-based schemes again provide the best figures. It should be noted that, in the computation of throughput we only considered the number of cycles spent for each message block. The initialization and finalization cycles and/or rounds were kept out of the calculation. In doing so, we assumed large data packets. However, for smaller data packets, these additional cycles should also be taken into account as they would considerably reduce the average throughput.
- **Power.** Again, permutation-based AE schemes have the best power figures as well as the lightweight block cipher based AE schemes. These figures also agree with their corresponding low areas.

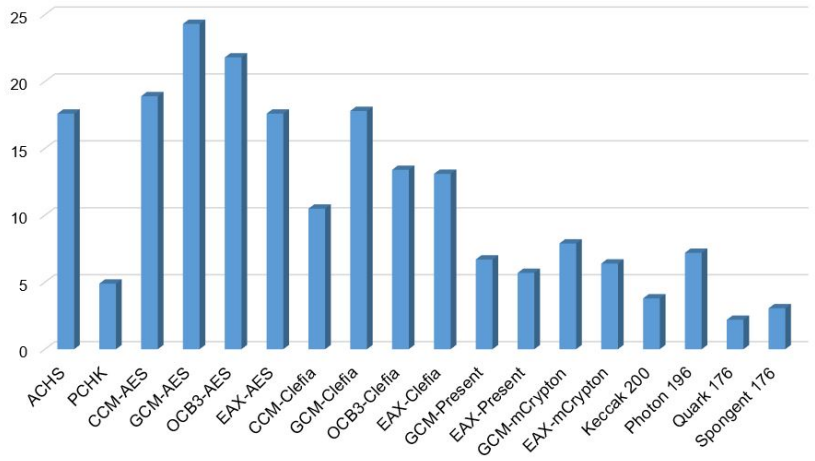

Fig. 8. Area results (kGE)

5 Conclusion

In this paper, we provide a comprehensive study on hardware, specifically ASIC, implementations of authenticated encryption schemes. We did not only evaluate *classical* encrypt-then-MAC schemes, but also more popular block cipher based AE schemes. In each case, we used different cipher combinations to highlight the impact of underlying cipher algorithm (and a hash function in case of encrypt-then-MAC). We furthermore extended our study to recently-introduced

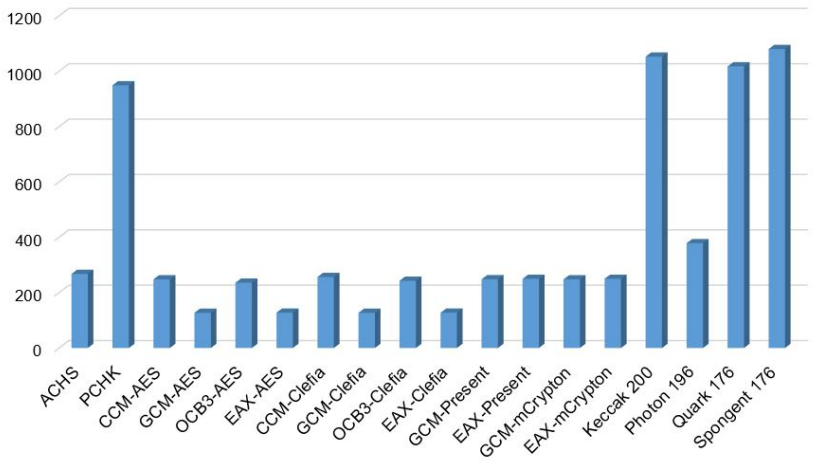

Fig. 9. Speed results (MHz)

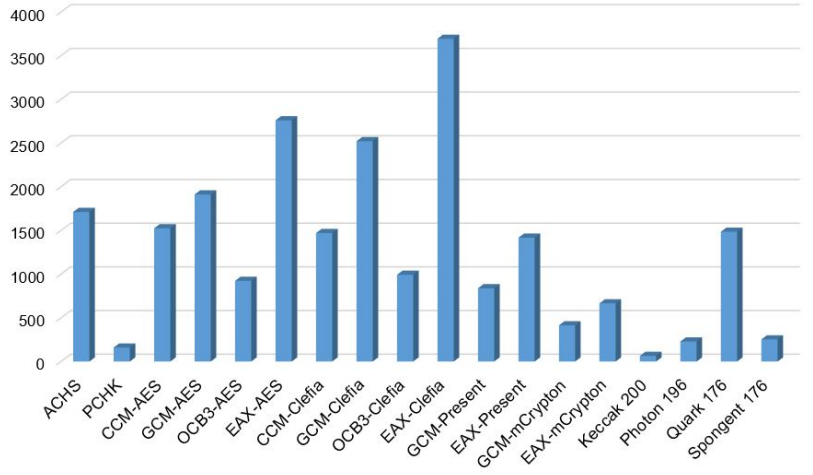

Fig. 10. Time × Area results (ns × kGE)

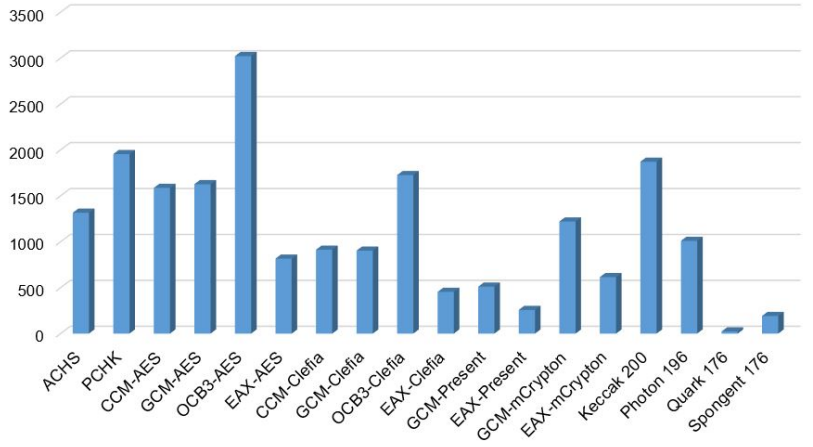

Fig. 11. Throughput results (Mbps)

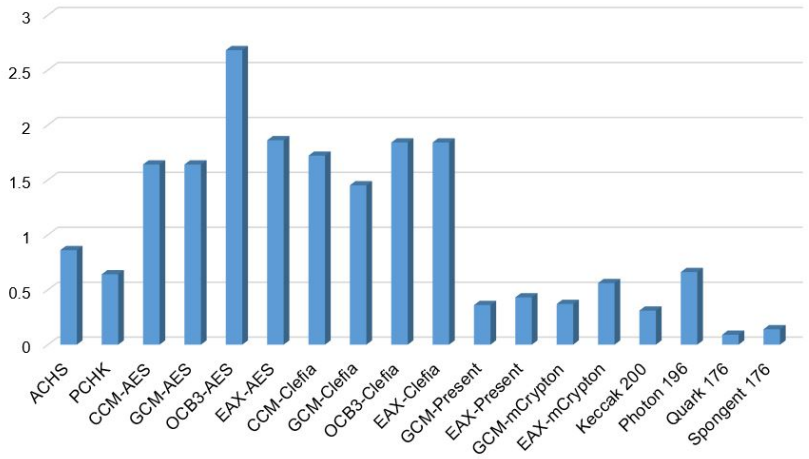

Fig. 12. Power results (mW)

permutation-based AE schemes, where we used different permutation functions for a fairer comparison.

To the best of our knowledge, this is the first comprehensive survey on different AE schemes from hardware (mainly ASIC) implementation point of view. Considering the wide range of underlying primitives used, it will provide a reference for anyone who is willing to incorporate an existing AE scheme in a new design or anyone who is willing to come up with a new AE scheme targeting a specific application.

From an ASIC designer's perspective, we have observed that simple schemes can provide lower gate counts and lower delays, thus result in higher overall performance without compromising security. It should also be noted that complex initialization (IV/counter generation) and finalization (tag generation) schemes should be avoided for hardware implementations. While such schemes have marginal effect on software implementations, their impact on hardware is considerably higher and worse. For example, the tweak initializations in OCB3 would only require additional initialization cycles in software. However, they would be bottlenecks for performance in a purely hardware-oriented implementation. Hence, we also assumed that they are precomputed in the upper protocol layer and provided to our modules from a lookup table. Otherwise, the favorable numbers of OCB3 would get considerably worse.

On the other hand, the newly-introduced permutation-based AE schemes show unprecedented performance on hardware, while they might suffer in a software implementation due to the software *unfriendliness* of the underlying permutation functions. However, this still remains to be seen, since no software evaluation has yet been performed on these schemes.

At first it seems fair to conclude that it is very hard to come with a scheme favorable and suitable for both hardware and software. However, the schemes with Keccak will soon probably contradict with this proposition. With its success in the SHA-3 competition, it is very likely that the next generation processors will incorporate some form of support for efficient software implementations of Keccak, from which Keccak-based AE schemes can also profit. We can summarize our finding as follows:

– **Avoid pre-processing:** Any type of pre-processing, i.e. key expansion, state initialization, etc., should be avoided. In practice, applications that require authenticated encryption, such as IP security, usually require processing of small to medium sized data blocks – a few bytes to 2 KB max. While core authenticated encryption process of such data blocks take relatively short time, any kind of pre-processing adds relatively high additional processing time, thereby

increasing the average time per byte of data. Furthermore, pre-processing usually require additional storage, which increases the overall hardware area. AE schemes with online processing capability should be favored.
- **Avoid complex operations:** Unlike software solutions, where the target platform, i.e. microprocessor, is capable of complex additions, subtractions, multiplications and even divisions of large data blocks within even a single clock cycle, custom hardware solutions suffer from such operations. Instead, simple permutation operations, which have little-to-no cost in hardware should be favored. Similarly, for confusion (substitution – s-box) operations, smaller block sizes (4 to 6-bits) should be selected. Larger blocks result in large lookup tables with higher hardware area costs.
- **Avoid heterogeneous operations:** Even if the underlying operations are simple (both area and time-efficient), diversified types of operations require different hardware blocks, resulting in additional area and hence increased time-area product. The best example is encrypt-then-MAC scheme, where even the simple primitives like PRESENT and Keccak are used, the cumulative block area, as well as the design and optimization effort, are doubled. The effect of avoiding this is clearly seen in the permutation based AE schemes, where a single functional block carries out both encryption and MAC operations without any additional hardware cost.
- **Respect software:** It should be always kept in mind that any AE should be capable of being operable on software platforms as well as hardware. Software aspects should not be completely dismissed. All the building blocks, modules, operations should also be checked for software performance. Compromises should be made where necessary, in order to find the right balance between hardware and software implementations.

In our opinion, the resulting cipher portfolio of the authenticated encryption competition should cover all these angles, and incorporate building blocks that are very easily implementable both on software and hardware. In that sense, permutation-based schemes with both hardware and software friendly permutation functions as well as AES-based schemes seem to be the most promising ones.

One must note that our conclusions are solely based on the chosen implementations and do not generalize beyond them. We do not claim covering all angles. However, we believe that our work will be more than instrumental for those who plan to deploy authenticated encryption in their work. We also realize that a complete hardware survey can only be possible with the inclusion of FPGA-based implementations. But then the task becomes much more complicated. In ASIC implementations, we suffered from limited number of configurations and parameters. In FPGA implementations, we would suffer not only from limited number of FPGA models from a specific vendor, but also from limited number of vendors. We would also have to consider different design choices, i.e. register-based vs block-memory-based, use vs lack of DSP blocks, and so on. It would therefore be prudent to perform an FPGA survey that focuses on a specific design target (i.e. low-cost, high-performance, etc.) and limits the vendor and model options. We leave the FPGA survey to another study – one that would possibly focus on the CAESAR finalists vs existing schemes.

Another future direction in this study would be to address the physical attacks. It becomes more and more important to implement counter measures for physical attacks. In several countries, implementation of side-channel attack counter measures have already become an obligation for certifiability of secure devices. We see side-channel attack resistant implementation of authenticated encryption as an important research area and a future direction for the continuation of this study.

References

1. AES. *Advanced Encryption Standard*. FIPS PUB 197, Federal Information Processing Standards Publication,

2001.
2. Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. QUARK: A Lightweight Hash. In *Cryptographic Hardware and Embedded Systems - CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2010.
3. Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer-Verlag, 2000.
4. Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations Among Notions and Analysis of the Generic Composition Paradigm. *Journal of Cryptology*, 21(4):469–491, 2008.
5. Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX Mode of Operation. In *Fast Software Encryption - FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer-Verlag, 2004.
6. Steven M. Bellovin. Problem Areas for the IP Security Protocols. In *USENIX Security Symposium 1996*, pages 205–214, 1996.
7. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *Keccak Specifications*, 2009.
8. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-pass Authenticated Encryption and Other Applications. In *Selected Areas in Cryptography - SAC 2011*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer-Verlag, 2012.
9. Begül Bilgin, Andrey Bogdanov, Miroslav Knezevic, Florian Mendel, and Qingju Wang. Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware. In *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 142–158. Springer-Verlag, 2013.
10. John Black and Hector Urtubia. Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption. In *USENIX Security Symposium 2002*, pages 327–338, 2002.
11. Andrey Bogdanov, Miroslav Knežević, Gregor Leander, Deniz Toz, Kerem Varıcı, and Ingrid Verbauwhede. SPONGENT: A Lightweight Hash Function. In *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 312–325. Springer-Verlag, 2011.
12. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and Charlotte Vikkelsø. PRESENT: An Ultra-Lightweight Block Cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer-Verlag, 2007.
13. Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen, and Elmar Tischhauser. ALE: AES-Based Lightweight Authenticated Encryption. In *Fast Software Encryption - FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2013.
14. CAESAR. Competition for Authenticated Encryption: Security, Applicability, and Robustness. `http://competitions.cr.yp.to/caesar.html`, 2013.
15. DES. *Data Encryption Standard*. FIPS PUB 46, Federal Information Processing Standards Publication, 1977.
16. Doug Whiting and Russ Housley and Niels Ferguson. *Counter with CBC-MAC (CCM)*. Internet Engineering Task Force (IETF) – RFC 3610 (Informational), 2003.
17. Thomas Eisenbarth, Sandeep Kumar, Christof Paar, Axel Poschmann, and Leif Uhsadel. A Survey of Lightweight-Cryptography Implementations. *IEEE Des. Test*, 24(6):522–533, 2007.
18. Daniel Engels, Markku-Juhani O. Saarinen, Peter Schweitzer, and Eric M. Smith. The Hummingbird-2 Lightweight Authenticated Encryption Algorithm. *IACR Cryptology ePrint Archive*, 2011:126, 2011.
19. Conrado P. L. Gouvêa and Julio López. High Speed Implementation of Authenticated Encryption for the MSP430X Microcontroller. In *Cryptology and Information Security in Latin America - LATINCRYPT 2012*, volume 7533 of *Lecture Notes in Computer Science*, pages 288–304. Springer-Verlag, 2012.
20. Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON Family of Lightweight Hash Functions. In *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer-Verlag, 2011.
21. HELION. AES-CCM cores. `http://www.heliontech.com/aes_ccm.htm`, 2014.
22. HELION. AES-GCM cores. `http://www.heliontech.com/aes_gcm.htm`, 2014.
23. IPSec. *Internet Protocol Security*. Internet Engineering Task Force (IETF) – RFC 4835, 2007.
24. ISO/IEC 19772:2009. *Information technology – Security techniques – Authenticated encryption*, 2013.
25. ISO/IEC 29192-2:2012. *Information technology – Security techniques – Lightweight cryptography – Part 2: Block ciphers*, 2012.
26. ISO/IEC 9797-1:2011. *Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms Using a Block Cipher*, 2011.
27. Sukumar Jairam, Madhusudan Rao, Jithendra Srinivas, Parimala Vishwanath, H. Udayakumar, and Jagdish C. Rao. Clock Gating for Power Optimization in ASIC Design Cycle Theory & Practice. In *International Symposium on Low Power Electronics and Design – ISLPED 2008*, pages 307–308. ACM, 2008.
28. Miroslav Knežević, Ventzislav Nikov, and Peter Rombouts. Low-Latency Encryption - Is "Lightweight = Light + Wait"? In *Cryptographic Hardware and Embedded Systems - CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 426–446. Springer-Verlag, 2012.

29. Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In *Fast Software Encryption - FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer-Verlag, 2011.
30. Ted Krovetz and Phillip Rogaway. *The OCB Authenticated-Encryption Algorithm*. Internet Engineering Task Force (IETF) – draft-krovetz-ocb-04, 2013.
31. Chae Lim and Tymur Korkishko. mCrypton – A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In *Information Security Applications 2006*, volume 3786 of *Lecture Notes in Computer Science*, pages 243–258. Springer-Verlag, 2006.
32. Helger Lipmaa, David Wagner, and Phillip Rogaway. *Comments to NIST Concerning AES Modes of Operation: CTR-Mode Encryption*, 2000.
33. David McGrew and Kenny Paterson. *Authenticated Encryption with AES-CBC and HMAC-SHA*. Internet Engineering Task Force (IETF) – draft-mcgrew-aead-aes-cbc-hmac-sha2-01, 2012.
34. David McGrew and John Viega. *The Galois/Counter Mode of Operation (GCM)*. NIST Modes Operation Symmetric Key Block Ciphers, 2005.
35. Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer-Verlag, 2011.
36. NANGATE. 45 nm Open Cell Library, 2008.
37. Dang Khoa Nguyen, Leonardo Lanante, and Hiroshi Ochi. High Throughput– Resource Saving Hardware Implementation of AES-CCM for Robust Security Network. *Journal of Automation and Control Engineering*, 1(3):250–254, 2013.
38. Milind M. Parelkar and Kris Gaj. Implementation of EAX Mode of Operation for FPGA Bitstream Encryption and Authentication. In *Proceedings of IEEE International Conference on Field-Programmable Technology*, 2005.
39. Phillip Rogaway, Mihir Bellare, and John Black. OCB: A Block-cipher Mode of Operation for Efficient Authenticated Encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.
40. Akashi Satoh, Takeshi Sugawara, and Takafumi Aoki. High-Performance Hardware Architectures for Galois Counter Mode. *IEEE Transactions on Computers*, 58(7):917–930, 2009.
41. SHA. *Secure Hash Standard*. FIPS PUB 180-2, Federal Information Processing Standards Publication, 2002.
42. SHA-3. Cryptographic Hash Algorithm Competition. NIST, 2007.
43. Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit Block Cipher CLEFIA (Extended Abstract). In *Fast Software Encryption - FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer-Verlag, 2007.
44. Antonio Giuseppe Maria Strollo, Ettore Napoli, and Davide De Caro. New Clock-gating Techniques for Low-power Flip-flops. In *International Symposium on Low Power Electronics and Design – ISLPED 2000*, pages 114–119. ACM, 2000.
45. The Internet Engineering Task Force. *RFC Index*, 2016. http://www.rfc-editor.org/rfc-index.html.
46. Tolga Yalçın and Elif Bilge Kavun. On the Implementation Aspects of Sponge-based Authenticated Encryption for Pervasive Devices. In *Smart Card Research and Advanced Applications - CARDIS 2012*, volume 7771 of *Lecture Notes in Computer Science*, pages 141–157. Springer-Verlag, 2013.

Table 1. Evaluation results

Type of Algorithm	**Area (kGE)**	**Speed (MHz)**	**Time×Area (ns×kGE)**	**Throughput (Mbps)**	**Power (mW)**	**Tput/Power (Mbps/mW)**
Encryption+Decryption Cores						
AES-128	16.1	268.60	601.1	3438.1	0.81	4244.5
Clefia-128	7.3	266.50	493.1	1895.1	0.45	4211.3
mCrypton-96	4.8	627.0	99.5	3086.7	0.23	13420.7
PRESENT-80	3.9	1007.0	122.11	2079.0	0.17	12229.2
Encrypt-then-MAC						
AES-128 SHA-256	17.6	267.52	1711.6	1316.2	0.86	1530.5
PRESENT-80 Keccak-200	4.9	948.76	160.3	1954.45	0.64	3053.8
Counter with CBC-MAC (CCM) Mode of Operation						
AES-128	18.9	247.89	1524.8	1586.5	1.64	967.4
Clefia-128	10.5	256.61	1471.3	913.53	1.72	531.1
AES-128[a]	**20.5**	**264.00**	**975.5**	**2690.0**	**N.A.**	**N.A.**
Galois/Counter Mode (GCM) of Operation						
AES-128	24.3	127.08	1912.2	1626.62	1.64	991.8
Clefia-128	17.8	127.08	2521.7	903.54	1.45	623.1
PRESENT-80	6.7	248.20	838.5	511.29	0.36	1420.3
mCrypton-96	7.9	248.02	414.3	1220.26	0.37	3298.0
AES-128[b]	**34.5**	**200.00**	**1725.0**	**2560.0**	**N.A.**	**N.A.**
Offset Codebook (OCB3) Mode of Operation						
AES-128	21.8	236.07	923.4	3021.7	2.68	1127.5
Clefia-128	13.4	242.78	993.6	1726.17	1.84	938.1
AES-128[c]	**5.9**	**200.00**	**6666.7**	**113.3**	**2.11**	**53.7**
EAX Mode of Operation						
AES-128	17.6	127.52	2760.4	816.13	1.86	438.8
Clefia-128	13.1	127.52	3693.7	453.97	1.84	246.7
PRESENT-80	5.7	249.88	1417.3	257.38	0.43	598.6
mCrypton-96	6.4	249.87	666.3	614.68	0.56	1097.6
Permutation-based Authenticated Encryption						
Keccak-200-64	3.8	1052.63	64.95	1871.34	0.31	6036.6
Photon-196-64	7.2	378.64	229.2	1009.71	0.66	1529.9
Quark-176-64	2.2	1017.30	1485.5	23.12	0.09	256.9
Spongent-176-64	3.06	1079.91	253.8	191.98	0.14	1371.3
Keccak-200-64[d]	**5.9**	**154.20**	**38.3**	**4934.4**	**6.47**	**762.9**

Note: The rows in boldface represent *state-of-the-art* implementations.

[37] – In this work, an AES-CCM core is implemented in parallel.

[40] – In this work, a GCM-AES circuit is implemented in serial.

[13] – In this work, OCB2 is implemented using a serialized AES design.

[46] – In this work, Keccak-200-64 is implemented in parallel (MonkeyDuplex construction).