FELICS-AEAD: Benchmarking of Lightweight Authenticated Encryption Algorithms

Luan Cardoso dos Santos(✉), Johann Großschädl, and Alex Biryukov

CSC and SnT, University of Luxembourg,
6, Avenue de la Fonte, 4364 Esch-sur-Alzette, Luxembourg
{luan.cardoso,johann.groszschaedl,alex.biryukov}@uni.lu

Abstract. Cryptographic algorithms that can simultaneously provide both encryption and authentication play an increasingly important role in modern security architectures and protocols (e.g. TLS v1.3). Dozens of authenticated encryption systems have been designed in the past five years, which has initiated a large body of research in cryptanalysis. The interest in authenticated encryption has further risen after the National Institute of Standards and Technology (NIST) announced an initiative to standardize "lightweight" authenticated ciphers and hash functions that are suitable for resource-constrained devices. However, while there already exist some cryptanalytic results on these recent designs, little is known about their performance, especially when they are executed on small 8, 16, and 32-bit microcontrollers. In this paper, we introduce an open-source benchmarking tool suite for a fair and consistent evaluation of Authenticated Encryption with Associated Data (AEAD) algorithms written in C or assembly language for 8-bit AVR, 16-bit MSP430, and 32-bit ARM Cortex-M3 platforms. The tool suite is an extension of the FELICS benchmarking framework and provides a new AEAD-specific low-level API that allows users to collect very fine-grained and detailed results for execution time, RAM consumption, and binary code size in a highly automated fashion. FELICS-AEAD comes with two pre-defined evaluation scenarios, which were developed to resemble security-critical operations commonly carried out by real IoT applications to ensure the benchmarks are meaningful in practice. We tested the AEAD tool suite using five authenticated encryption algorithms, namely AES-GCM and the CAESAR candidates ACORN, ASCON, Ketje-Jr, and NORX, and present some preliminary results.

Keywords: Internet of Things · Lightweight cryptography · Authenticated Encryption · Application Program Interface · Evaluation scenario

1 Introduction

An Authenticated Encryption (AE) algorithm can be loosely defined as a symmetric cryptographic algorithm that is capable to (simultaneously) assure the

S. Belaïd and T. Güneysu (Eds.): CARDIS 2019, LNCS 11833, pp. 216–233, 2020.
https://doi.org/10.1007/978-3-030-42068-0_13

confidentiality *and* authenticity of data [3,11]. A special form of AE, known as *Authenticated Encryption with Associated Data (AEAD)*, allows a part of the data to remain unencrypted, while still all data gets authenticated. The notion of AEAD was first formalized by Rogaway [14] in 2002 and has applications in such areas as network packet encryption where the header (which contains the destination address) needs to be readable by routers, but should nonetheless be authenticated and integrity-protected. An AEAD algorithm takes a quadruple of the form (M, A, K, N) as input and outputs a tuple (C, T), where M is the message to be encrypted and authenticated, A is the associated data that gets authenticated only (but not encrypted), K is the secret key, N is a nonce, C is the ciphertext, and T is an authentication tag. Conversely, the decryption uses (C, A, K, N, T) as input and outputs the original message M if T is valid, or an error symbol $\perp$ otherwise. The two essential security goals an AEAD algorithm has to achieve are confidentiality and authenticity; a mathematically rigorous definition of both was given by Rogaway [14]. Informally, confidentiality means that a passive adversary with access to C and T should not be able to deduce any information about M, except of its length. Authenticity generally refers to the ability to thwart forgery attacks, which means an active adversary should have a very low success probability when attempting to fabricate a (C, T)-tuple that the decrypting party will verify as authentic.

Initially, AEAD schemes were created by combining a block cipher in some mode of operation with a Message Authentication Code (MAC) algorithm. A clear disadvantage of this approach is the necessity of having two different primitives and requiring two passes over the message. Modern constructions use a different approach, where a single algorithm is able to deliver authenticated encryption, with a single pass over the message. In recent years, the cryptographic community has shown great interest in AEAD because of the CAESAR competition and the NIST call for lightweight primitives. CAESAR (short for Competition for Authenticated Encryption: Security, Applicability, and Robustness) is an already finished competition whose objective was to select a portfolio of AEAD algorithms. It followed the spirit of previous cryptographic competitions, such as the one that yielded the now omnipresent block cipher AES. In 2018, the NIST officially announced the initiation of a process to solicit, evaluate, and standardize lightweight cryptographic algorithms—namely AEAD schemes and hash functions—that are suitable for constrained environments where the current standards can not provide acceptable performance. The motivation behind this initiative is the emergence of more and more application domains where constrained devices are interconnected to form the so-called Internet of Things (IoT). Security and privacy are extremely important in the IoT, but cannot always be provided by the currently standardized cryptosystems. This is because the severe constraints under which present (and future) IoT devices are expected to operate were not anticipated 20–25 years ago when many of the current NIST standards (e.g. AES, SHA-2) were designed.

Motivation and Research Needs. In response to NIST's call for proposals for lightweight AEAD algorithms and hash functions, a total of 57 candidates were submitted by March 29, 2019. These candidates are currently evaluated in an open process taking various criteria into account, which include besides security (i.e. resistance against known cryptanalytic attacks) also practical aspects like performance and resource requirements (e.g. silicon area, memory footprint, code size) when implemented in hardware and software [13]. The NIST anticipates an initial (i.e. first-round) evaluation period of about six months to filter out candidates with obvious weaknesses and narrow the candidate pool for a more careful study and analysis in a second round. In total, the NIST estimates a duration of two to four years until the publication of a first draft standard and emphasizes that "the success of the lightweight crypto standardization process relies on the efforts of the researchers from the cryptographic community that provide security, implementation, and performance analysis of the candidates"[1]. Most papers introducing a new AEAD algorithm report some kind of results of some kind of performance evaluation on some kind of platform using some kind of implementation. Unfortunately, these results are usually not suitable for a comparison of the efficiency of two or more algorithms since it is not easily possible to take differences in the characteristics of the target platforms or differences in the simulation/measurement conditions into account. There is a need for a way to compare performance figures for many algorithms consistently and fairly so that designers and implementers of IoT applications can make better decisions regarding which algorithm is the most suitable one under a given set of efficiency requirements and resource constraints.

In the course of the CAESAR competition, the eBACS framework [4] was used for the bechmarking of the submitted AEAD algorithms. However, the original eBACS tools only support 64-bit Intel/AMD processors and high-end ARM models, mostly from the Cortex-A series, whereas many IoT devices are equipped with low-end microcontrollers, e.g. 8-bit AVR ATmega, 16-bit TI MSP430, or 32-bit ARM Cortex-M. These microcontrollers are optimized for small silicon area and low power consumption, which means they have totally different characteristics than their 64-bit counterparts. These differences manifest not only in the word size, but also the instruction set, the size of the register file, the latency of individual instructions, the degree of instruction-level parallelism, and many other aspects. For example, 64-bit Intel or ARM processors have a register space of 128 bytes (or even more when taking vector registers into account), whereas the MSP430 platform (which lies at the opposite end of the spectrum) provides 24 bytes altogether. Furthermore, most 8 and 16-bit microcontrollers can only execute shifts or rotations at a rate of one bit per cycle, whereas more powerful processors are capable to perform n-bit shifts/rotations in a single cycle. For all these reasons, benchmarking results generated with eBACS are of little use when it comes to the evaluation of AEAD algorithms on microcontrollers.

[1] See https://csrc.nist.gov/projects/lightweight-cryptography/round-1-candidates (accessed 2019-07-15).

Aims and Contributions of This Paper. The present paper addresses the research needs identified above and puts forward a proposal for the benchmarking of lightweight AEAD algorithms. Our proposal aims to answer two basic questions that generally arise in the context of software benchmarking of cryptographic algorithms. The first question relates to the Application Program Interface (API) that implementations of a candidate algorithms have to follow to ensure a fair and consistent evaluation. We will argue in Subsect. 2.2 that, for the purpose of benchmarking, it makes sense to use a low-level API sense since it allows one to obtain more fine-grained results compared to a high-level API consisting of just the functions `encrypt` and `decrypt`. Furthermore, we introduce an API containing seven low-level functions, which we consider well suited for the benchmarking of AEAD algorithms. The second issue concerns the question of how to measure the execution time and other metrics of interest, which includes aspects like the length of the message M and the length of the associated data A. More concretely, how should the length-ratio of M and A be to get meaningful results? We will try to answer these questions in Subsect. 2.1 through the definition of so-called evaluation scenarios that aim to mimic security-related operations commonly carried out by "real" IoT devices. More concretely, these scenarios are inspired by the need for AEAD operations in two networking protocols with relevance for the IoT, namely IEEE 802.15.4 (the most common PHY/MAC-layer protocol for low-rate wireless networks) and IPv6.

We implemented both the low-level API for AEAD and the evaluation scenarios in the form of an extension to the well-known and widely-used FELICS (Fair Evaluation of Lightweight Cryptographic Systems) framework [7]. FELICS was originally created to support the collection benchmarking results for (lightweight) block ciphers on three embedded platforms: 8-bit AVR, 16-bit MSP430, and 32-bit ARM Cortex-M3. The full source code of FELICS is available under GPLv3 to increase the transparency and reproducibility of benchmarking results. Besides execution time, FELICS is also capable to determine the binary size and RAM footprint on the three currently supported platforms. The framework is modular, built on well documented and free compilers and tools, which allows easy extension of functionality and integration of new microcontroller platforms and evaluation scenarios. We tested the extended FELICS toolsuite using optimized C implementations of five AEAD algorithms (namely AES-GCM, ACORN, ASCON, Ketje-Jr, and NORX) that adhere to our low-level API. These tests confirm that FELICS-AEAD works properly and is able to collect large amounts of benchmarking results in an efficient and highly-automated fashion. An analysis of the collected benchmarking results for these five algorithms allows us to draw some conclusions about how basic design decisions like the organization of the "state" (i.e. whether the state is processed at a granularity of 32-bit words or 64-bit words) affect the performance on small microcontrollers.

2 The FELICS Framework and Its AEAD Extension

FELICS – ***F****air* ***E****valuation of* ***Li****ghtweight* ***C****ryptographic* ***S****ystems* – is a free and open source framework that assesses the efficiency of C and assembly implementations of lightweight cryptographic primitives on embedded devices. Following a modular design philosophy, the framework can easily be extended to accommodate new metrics, evaluation scenarios, and devices. FELICS is the core of an effort to increase the transparency in the analysis of lightweight algorithms' performances and aims to facilitate a fair comparison of a large number of candidates. Figure 1 gives an overview of the structure and main components of the FELICS framework.

2.1 Overview of Modules

FELICS is written in C, but also includes Bash and Python scripts. The framework was designed to work on Linux and allows the benchmarking of C and assembly implementations of cryptographic primitives that follow a set of predefined requirements. C was chosen because of its continuing popularity in the IoT and the fact that most reference implementations are written in this language. Furthermore, C code is highly portable, which is an important asset since there is no single dominating platform in the IoT. However, FELICS also supports the benchmarking of platform specific Assembler implementations to eliminate the impact of the compiler's ability (or inability) for code optimization. Hand-crafted Assembler code can take architecture-specific optimizations into account and has the potential to significantly outperform compiled C code.

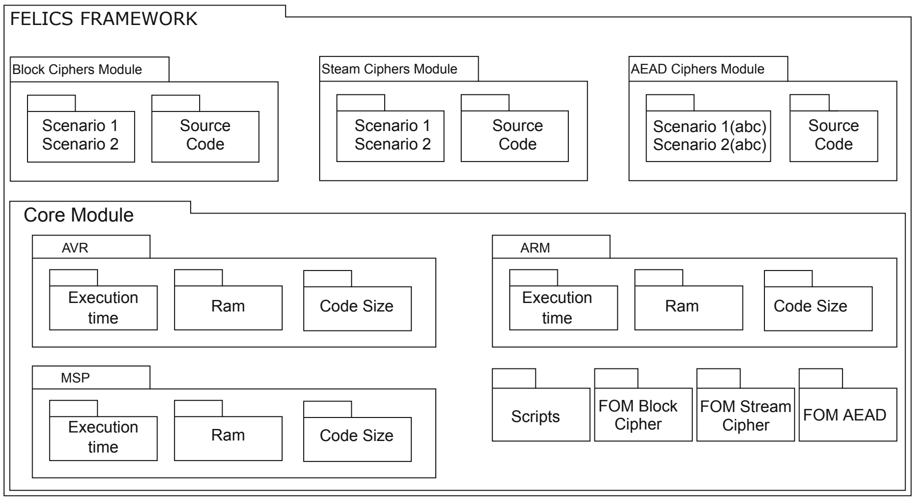

Fig. 1. Modular structure of the FELICS benchmarking framework.

Core Module. The Core module, as the name implies, is the main part of the framework, and provides the tools necessary to collect the metrics for each of the supported devices. This module aims to facilitate the integration of new target devices and new metrics. Collection of metrics can be done individually or in batch mode. Beyond metrics collection, the Core also defines modules to debug and evaluate ciphers in a PC, mainly to aid in the implementation and integration process of new ciphers by the framework's users. A Python script for processing the generated CSV files and to assemble a ranking of candidates based on a so-called Figure-Of-Merit (FOM) is also present (see [8] for details).

Authenticated Encryption Module. This module allows the evaluation of lightweight AEAD ciphers. To allow the framework to extract the metrics, each cipher's implementation must follow the defined API.

A template for implementation, as well as implementations of identity ciphers, are provided with the module and can be used as a guide to help new users to integrate new implementations. The complete rules and step-by-step integration guide for cipher implementations can be found in the `README` file in the example cipher.

The framework supports cipher evaluation based on scenarios. Scenarios implement common real-world use cases, with practical relevance for IoT, with the main objective of generating realistic benchmark results that are meaningful in the real world. The current scenarios in the AEAD module of FELICS are divided into three main groups:

– **Debug and verification Scenario:** Also called Scenario 0, is mainly used for debugging purposes. It operates over a single block of input and allows the implementers to check their implementations on known test vectors.
– **IEEE 802.15.4 Scenarios:** These scenarios are based on the security needs of data communication in wireless sensor networks and other IoT applications using the IEEE 802.15.4 MAC/PHY-layer protocol. The maximum frame size of IEEE 802.15.4 is 127 bytes; the length of the header depends on various factors, such as the format of the source and destination addresses, but can not exceed 25 bytes. This leaves (at least) 102 bytes as frame payload. IEEE 802.15.4 supports three kinds of security services, namely (i) "Encryption Only" with AES in counter mode, (ii) "Authentication Only" with AES-CBC-MAC producing a MAC of either 32, 64, or 128 bits, and (iii) "Authenticated Encryption" using AES-CCM with the same MAC lengths.
  - **Scenario 1a:** Encryption of 102 bytes of data.
  - **Scenario 1b:** Authentication of 86 bytes of payload and 25 bytes of header. This scenario assumes that 16 bytes of payload are reserved to write the authentication tag.
  - **Scenario 1c:** Authenticated encryption of 86 bytes of payload and 25 bytes of header (which is authenticated but not encrypted). As with Scenario 1b, the authentication tag has a length of 16 bytes.
– **IPv6 Scenarios:** These scenarios are based on the use cases of IPv6 frames, as defined in RFC 2460. The MTU of IPv6 is at least 1280 bytes and the

header has a fixed length of 40 bytes. Based on experiments with the Network Simulator NS-3, we found that the following message and associated data lengths serve as good representatives for real-world scenarios.

- **Scenario 2a:** Encryption of 1240 bytes of data.
- **Scenario 2b:** Authentication of 1224 bytes of payload and 40 bytes of header.
- **Scenario 2c:** Authenticated encryption of 1224 bytes of payload and 40 bytes of header.

The IEEE 802.15.4 and IPv6 scenarios differ not only in the amount of data to be protected (127 bytes vs 1280 bytes), but also in the relation of data-length of AD-length. In the former case, the AD/D ratio is 0.29, whereas in the latter case the AD-length is negligible in relation to the D-length.

2.2 API for Authenticated Encryption

The FELICS API aims to offer a generic and well-specified interface for the most common operations performed by an AEAD algorithm. Different from other frameworks, the FELICS API is composed of seven low-level functions. While this may introduce difficulties for certain implementation techniques (e.g. bitslicing), the low-level API gives the framework more flexibility and allows one to obtain more fine-grained benchmarking results. Such fine-grained results can be useful, for example, when one wants to analyze *why* a given AEAD algorithm is more or less efficient and its competitors. Our seven functions are described below and their prototypes are given in Listing 1.

- `Initialize`: This function receives as parameters pointers to the algorithm's state, key, and nonce, and should execute the cipher's initialization procedures.
- `ProcessAssocData`: This function receives as parameters a pointer to the state, a byte stream of associated data, as well as its length.
- `ProcessPlaintext`: This function receives as parameters a pointer to the state, a byte stream of data, as well as the length of plaintext and ciphertext. The ciphertext should overwrite the plaintext.
- `ProcessCiphertext`: This function receives as parameters a pointer to the state, a byte stream of data, as well as the length of plaintext and ciphertext. The plaintext should overwrite the ciphertext.
- `Finalize`: This function receives as parameters pointers to the state and key, and executes the finalization steps on the internal state, preparing it for the authentication tag generation.
- `GenerateTag`: This functions receives as parameters a pointer to the internal state and the authentication tag and should write the appropriate information on the authentication tag.
- `VerifyTag`: This function received two pointers to authentication tags, and compare both. Returns `(int)(1)` if the tags match, and `(int)(0)` otherwise.

Listing 1. Function prototypes of the low-level AEAD API.

```
void Initialize(uint8_t *state, const uint8_t *key,
     const uint8_t *nonce);
void ProcessAssocData(uint8_t *state, uint8_t *assocData,
     size_t assocDataLen);
void ProcessPlaintext(uint8_t *state, uint8_t *message,
     size_t messageLen);
void ProcessCiphertext(uint8_t *state, uint8_t *message,
     size_t messageLen);
void Finalize(uint8_t *state, uint8_t *key);
void GenerateTag(uint8_t *state, uint8_t *tag);
int  VerifyTag(uint8_t *state, uint8_t *tag);
```

NIST specified a high-level API consisting of two functions (namely `aead_encrypt` and `aead_decrypt`), which submitters of AEAD candidates had to follow when they developed the (mandatory) reference implementation and an (optional) optimized implementation. While such a high-level API is convenient for software developers using AEAD algorithms, it is not necessarily a good choice for collecting benchmarking results, especially in Scenario 0. This is probably best explained taking the block-cipher benchmarks from [9] as example. Similar to AEAD, one can benchmark block ciphers using either a high-level or a low-level API. The former consists of generic functions for encrypting/decrypting of an arbitrary amount of data using a specified mode operation. On the other hand, the low-level API consists of two functions for each encryption and decryption, one to encrypt/decrypt a single block, and one to perform the encrytion/decryption key schedule. In order to minimize the overall development effort, the high-level functions can simply be implemented as wrappers over the low-level functions. However, using the low-level API for benchmarking in Scenario 0 makes certain properties of ciphers more apparent than the high-level API. For example, RC5 is extremely fast, but has a very costly key schedule, which becomes immediately evident with benchmarking results obtained with the low-level API. Therefore, RC5 is unattractive for scenarios where the amount of data to be encrypted or decrypted is small. This information is not so directly obvious when benchmarking results are generated with the high-level API.

2.3 Target Devices and Evaluation Metrics

For this framework, three widely used microcontrollers were chosen as representatives of the most used 8, 16, and 32-bit platforms used in the IoT. These microcontrollers have been optimized for small area and low power consumption. Their main characteristics are summarized in Table 1 and a brief description of each will follow on the next paragraphs.

The **AVR ATMega 128** is a microcontroller manufactured by Atmel, featuring 32 8-bit registers (`R0 - R31`) with single clock access time. Six of those

Table 1. Key characteristics of the target microcontrollers.

Characteristic	AVR	MSP	ARM
CPU	8-bit RISC	16-bit RISC	32-bit RISC
Frequency	16 MHz	8 MHz	84 MHz
Registers	32	16	21
Architecture	Harvard	Von Neumann	Havard
Flash	128 KB	48 KB	512 KB
SRAM	4 KB	10 KB	96 KB
Supply voltage	4.6–5.5 V	1.8–3.6 V	1.6–3.6 V

registers can also be used as 16-bit indirect address pointers for data space. The instructions are executed within a two-stage, single-level pipeline, with most of its 133 instructions requiring a single cycle to execute. AVR processors are based on a modified Harvard architecture, where program and data are stored in separate physical memory regions in different physical addresses. Regarding memory, the ATmega128 comes with 128 KB Flash amd 4 KB SRAM.

The **MSP430F1611** microcontroller is a RISC CPU produced by Texas Instruments. It follows a Von Neumann architecture, and features 16 registers, with 12 being general purpose. Operations over registers take one clock cycle, while the other instructions depend on its format and addressing mode used. Memory wise, the MSP430 has one shared address space for special function registers, peripherals, RAM and FLASH. It has 48 KB of Flash and 10 KB of SRAM. Typical applications include medical devices and smart meters.

The 32-bit **Atmel SAM3X8 Cortex M3** is a RISC CPU that executes the Thumb-2 instruction set. This processor has a three-level pipeline and 13 general-purpose registers. It features 512 KB of Flash and 96 KB of SRAM divided into two banks of 64 KB and 32 KB. The Cortex-M3 is specially designed to achieve high performance in power-sensitive embedded applications, such as microcontrollers, automotive and industrial controllers, wireless networking, and others. This processor runs at a maximum frequency of 84 MHz.

For cipher evaluation, three metrics are used: Execution time, RAM usage, and code size. These metrics were chosen because they outline the main characteristics of the implementations. Secondary metrics, such as energy consumption were not included mainly due to being closely related to the basic metrics.

Execution time consists in measuring the number of cycles necessary to execute a given operation. This metric is extracted by using either a cycle-accurate simulator a development board. Extraction of cycle-counter uses AVRORA [15] for the AVR processor, and `MSPDebug` [2] for MSP. Extraction of cycle counter on ARM is done via the automatic insertion of code to read ARM's system time registers. One important detail regarding ARM's measurements is that there may exist variations in the extracted numbers, due to different instructions being generated at compilation time and memory alignment of test data.

RAM consumption is a combination of stack and data requirements. The stack consumption describes the maximum amount of RAM used to store local variables and return addresses after interruptions and system calls. The data requirement represents the static RAM usage and is given by the size of the constants stored in the device's RAM. Static RAM consumption is measured using the GNU `size` tool. The stack consumption is measured using a `gdb` client and the target simulator or development board.

Code size is measured in bytes and quantifies the amount of storage an operation or evaluation scenario occupies in the non-volatile memory of the target device. It is measured using the GNU `size` tool on the appropriate object files. To obtain the overall code size, the framework simply sums the size of the `text` and `data` sections, which contain, respectively, the executable instructions generated by the compiler and the static variables that are initialized with a non-zero value.

Figure of Merit. Due to space limitations, normally only a subset of data can be correctly shown in publications. To aid in the classifications of the evaluated ciphers, FELICS introduces the *Figure-of-Merit* (FOM), that can be used to rank the analyzed ciphers. For each implementation i and platform d, a performance indicator p_{i_d} that aggregates the metrics from $M = \{$execution time, RAM consumption, code size$\}$ as

$$p_{i,d} = \sum_{m \in M} w_m \frac{v_{i,d,m}}{\min_i(v_{i,d,m})}$$

where $v_{i,d,m}$ is the value of the metric m for the implementation i on the platform p; and w_m is the relative weight for the metric m, with $w_m = 1$ by default for all platforms. Then, for each cipher and the selected set of best implementations i_{AVR}, i_{MSP}, and i_{ARM} (one for each platform) the FOM is calculated as the average performance indicator across the three platforms:

$$\mathrm{FOM}(i_{AVR}, i_{MSP}, i_{ARM}) = \frac{p_{i_{AVR}} + p_{i_{MSP}} + p_{i_{ARM}}}{3}$$

3 Analyzed AEAD Algorithms

In this section, we briefly describe the ciphers implemented in FELICS, as an example and initial work for the framework. These ciphers were chosen for their relevance in the context of IoT and lightweight cryptography, as well for being part of an ongoing effort of standardizing AEAD schemes.

ACORN. Acorn is an AEAD scheme created by Hongjun Wu, and finalist of the CAESAR competition. It features a stream-cipher-like construction based on six concatenated linear feedback shift registers. The cipher's design benefits lightweight hardware implementations since the processing can be done in a bitwise fashion [17].

Table 2. Parameters of the evaluated ciphers, in bits.

Cipher	Block	Key	Nonce	State	Tag
NORX	384	128	128	512	128
ACORN	1	128	128	293	128
Ketje-Jr	16	128	48	200	128
ASCON	64	128	128	320	128
AES-GCM	128	128	96	1824	128

AES-GCM. The Galois/Counter mode is a mode of operation for 128-bit block ciphers, widely used together with the AES block cipher for its efficiency and performance. GCM is used in MACSec Ethernet Security, IEEE 802.11ad wireless protocols, Fibre Channel security protocols, and is also included in the NSA Suite B Cryptography, as well as various other software [12].

ASCON. Ascon is a family of AEAD ciphers, finalist of the CAESAR competition. It was designed by Christoph Dobraunig et al. in 2014. The main goal of ASCON is to achieve a very low memory footprint, both in hardware and software implementations, and still provide an adequate combination of security, speed, and size, with focus on the last. ASCON is based on the Sponge Design, being similar to SpongeWrap and MonkeyDuplex constructions [10].

Ketje. Ketje is a family of four AEAD algorithms, aimed to memory-constrained devices and that strongly relies on nonce uniqueness for security. It was designed by Guido Bertoni et al. and is a third-round candidate of the CAESAR competition. Ketje is based on a reduced round version of Keccak, over a MonkeyDuplex and MonkeyWrap constructions [5].

NORX. NORX is a family of AEAD ciphers created by Jean-Philippe Aumasson et al. in 2014. NORX supports associated data both as header and trailer. The algorithm also supports arbitrary parallelism in the payload processing step and is optimized for hardware and software implementations, with a specially SIMD friendly construction. NORX is based on ChaCha's permutation, with the integer addition replaced by an ARX approximation, which –according to the designers– allows simplified cryptanalysis and improves hardware efficiency [1].

4 Preliminary Results

Using the FELICS extension for authenticated encryption described in Sect. 2, we benchmarked optimized C implementations of the five AEAD algorithms on three platforms and for two evaluation scenarios plus Scenario 0, which is mainly

Table 3. Results for Scenario 1 (IEEE 802.15.4). For each platform and each cipher, the best implementation results are reported. The code size and memory consumption are specified for the whole scenario (and not just the AEAD algorithm alone), which includes the 127-byte IEEE 802.15.4 frame to be encrypted and/or authenticated. The smaller the Figure-of-merit, the better is the implementation of a cipher.

Cipher		AVR			MSP			ARM			FOM
		Size	Mem	Time	Size	Mem	Time	Size	Mem	Time	
NORX	S1a	4702	214	135640	3992	214	66738	1474	214	17227	4.3
	S1b	3936	223	90728	3482	223	53035	1148	223	10089	4.0
	S1c	5028	207	124062	4216	207	75727	1634	207	16685	4.5
ASCON	S1a	3734	190	519420	5656	190	599643	1712	190	80316	9.4
	S1b	3734	199	340671	5656	199	395564	1712	199	52958	8.9
	S1c	3734	183	534908	5656	183	619523	1712	183	83118	9.4
Ketje-Jr	S1a	5156	165	290446	6248	165	346867	3564	165	138867	9.4
	S1b	5156	174	211749	6248	174	254923	3564	174	99490	9.8
	S1c	5156	158	311949	6248	158	372720	3564	158	148381	9.7
ACORN	S1a	3292	191	337818	3170	191	456972	1954	191	191869	10.0
	S1b	3292	200	408914	3170	200	551501	1954	200	236235	15.7
	S1c	3292	184	464381	3170	184	626192	1954	184	267168	12.5
AES-GCM	S1a	6578	374	889573	6798	374	2137251	6096	374	1086449	41.5
	S1b	5944	383	447505	6782	383	1150450	6028	383	565606	34.0
	S1c	6578	367	975184	6798	367	2369572	6096	367	1197073	44.6

for debugging and verification. Table 2 summarizes the main characteristics of the specific variants of the AEAD algorithms we implemented.

The FELICS framework allows ranking all these implementations according to their execution time, RAM footprint, or code size in any scenario on any platform. Table 3 summarizes the results of Scenario 1, which is inspired by the need for security in the IEEE 802.15.4 protocol. This scenario actually consists of three sub-scenarios with different operations and slightly different lengths of the data to be encrypted and/or authenticated. However, all three sub-scenarios have in common that the amount of data is relatively small, namely between 86 and 102 bytes, due to the 127-byte MTU – maximum transmission unit – of the IEEE 802.15.4 protocol. If associated data is processed, its length is roughly one fourth of the data-length. Concretely, in Sub-scenario 1a ("encryption only"), 102 bytes of data are encrypted, whereas in Sub-scenario 1b ("authentication only") the size of the data is 86 bytes and the size of the associated data is 25 bytes. Finally, in Scenario 1c ("authenticated encryption") 86 bytes of data are encrypted and $86 + 25 = 111$ bytes are authenticated. NORX is the clear winner in all three sub-scenarios, followed by ASCON and Ketje-Jr, which perform very similar in all three sub-scenarios. However, the FOM score of the latter two algorithms is more than twice higher than that of NORX.

Table 4. Results for Scenario 2 (IPv6). For each platform and each cipher, the best implementation results are reported. The code size and memory consumption are specified for the whole scenario (and not just the AEAD algorithm alone), which includes the 1280-byte IPv6 packet to be encrypted and/or authenticated. The smaller the Figure-of-merit, the better is the implementation of a cipher.

Cipher		AVR			MSP			ARM			FOM
		Size	Mem	Time	Size	Mem	Time	Size	Mem	Time	
NORX	S2a	4702	1376	800313	3992	1376	501290	1474	1376	109933	4.1
	S2b	3936	1376	424601	3482	1376	246263	1148	1376	46113	3.7
	S2c	5028	1376	814467	4216	1376	508728	1634	1376	111361	4.2
ASCON	S2a	3292	1353	1811457	3170	1353	2454962	1954	1353	1013715	8.5
	S2b	3292	1353	1136110	3170	1353	1541295	1954	1353	644411	10.5
	S2c	3292	1353	1916720	3170	1353	2595469	1954	1353	1077068	8.7
Ketje-Jr	S2a	5156	1327	3026956	6248	1327	3623707	3564	1327	1481660	12.6
	S2b	5156	1327	1527941	6248	1327	1860262	3564	1327	751536	13.3
	S2c	5156	1327	3007966	6248	1327	3601416	3564	1327	1471405	12.5
ACORN	S2a	3734	1352	6174633	5656	1352	7109127	1712	1352	947367	13.9
	S2b	3734	1352	3146041	5656	1352	3619665	1712	1352	479574	14.2
	S2c	3734	1352	6112583	5656	1352	7039689	1712	1352	938358	13.6
AES-GCM	S2a	6578	1536	9807655	6798	1536	23748153	6096	1536	12036393	64.4
	S2b	5944	1536	3526008	6782	1536	9531538	6028	1536	4564667	54.2
	S2c	6578	1536	9812008	6798	1536	23796554	6096	1536	12050336	63.6

Finally, Table 4 shows the results of Scenario 2, which deals with security for the IPv6 protocol. This scenario is again split into three sub-scenarios, similar to the sub-scenarios in the context of IEEE 802.15.4 described above. However, the amount of data to be encrypted is much larger, around 1200 bytes, while the amount of associated data is relatively small; more concretely, the ratio between data and associated data is roughly 30:1. Again, NORX is the clear winner in all three sub-scenarios, followed by ASCON and Ketje-Jr. However, compared to the IEEE 802.15.4 scenarios, the difference between ASCON and Ketje-Jr is much bigger. Similar to before, the FOM score of NORX is significantly better than that of the runner-up ASCON.

It is interesting to observe that NORX is in both scenarios speed-wise much better than the other candidates. NORX outperforms its CAESAR competitors by a factor of at least two; in some extreme cases, NORX is even five times faster than the second-best algorithm. This significant difference begs for more analysis and raises the question of what design decisions make an AEAD algorithm efficient (or inefficient) on small microcontroller platforms. However, this question is difficult to answer since the efficiency of AEAD designs depends on many different factors, some of which are architecture-independent, i.e. affect the performance on 8, 16, 32, and 64-bit platforms similarly, whereas others are architecture-dependent in the sense that they impact the performance across platforms differently. An example of the latter is the organization of the state, i.e. whether the state is processed at a granularity of 32-bit words or 64-bit words. The benchmarked version of NORX processes the state in 32-bit words, whereas ASCON, ACORN, and Ketje-Jr operate on 64-bit words. Organizing the state in 64-bit quantities is the natural choice for designs aiming at high performance on Intel/AMD and 64-bit ARM processors as it allows one to exploit the full word-size of these processors, but may lead to suboptimal performance on smaller microcontroller platforms, which is due to three reasons.

First, C compilers for 8-bit AVR and 16-bit MSP microcontrollers (e.g. `mspgcc`) are, in general, not very good at handling 64-bit words (i.e. operands of type `uint64_t`). We assume this is because outside cryptography there are very few application domains where a programmer really needs a 64-bit integer on an 8 or 16-bit microcontroller. NORX128 uses 32-bit words, which seems to make it much easier for a C compiler to generate efficient code than for the other CAESAR candidates that process 64-bit words. The second reason is the small register space of 8 and 16-bit microcontrollers. For example, the MSP430 architecture comes with only twelve 16-bit general-purpose registers, which means it would theoretically be possible to hold three 64-bit words in the register file. However, in practice, this is not the case since always one or two registers are needed for temporary results and often also one register has to be set to 0. Therefore, it can be expected that no more than two 64-bit words can be kept in registers at any time, but it may be possible to accommodate five 32-bit words when the cipher's state is organized in 32-bit words. Finally, the third reason why 64-bit words can entail suboptimal performance is ARM-specific and relates to the fact that one of the two operands of an arithmetic/logical instruction is fed through a barrel-shifter before it enters the ALU, which means shifts and rotations can be executed "for free" together with other instructions. However, on a 32-bit ARM microcontroller, shifts and rotations are only free for 32-bit operands, but not for 64-bit quantities.

5 Comparison with Other Benchmarking Tools

Besides FELICS, there exist a few other tools for the benchmarking of cryptographic algorithms, of which eBACS and XXBX are the most closely related ones. eBACS (short for ECRYPT Benchmarking of Cryptographic Systems) was developed during the ECRYPT II project to evaluate the performance of cryptographic algorithms on Intel/AMD processors and high-end ARM models capable to run Linux (e.g. the Cortex-A series). It features modules for measuring the performance of public-key cryptosystems (called eBATS), stream ciphers (eBASC), hash functions (eBASH), and authenticated encryption algorithms (eBAEAD). Those modules operate all under a common framework called SUPERCOP (System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives) that allows benchmarking of C, C++ and assembly implementations. It comes with a large collection of implementations of cryptographic algorithms and automatically compiles source code using different compilers and compiler options. The execution time is extracted via a cycle counter (accessed through assembler code) for many different lengths of input data. Since execution time is the only metric measured by this framework, implementations are optimized solely for speed.

The eXternal Benchmarking eXtension [16] is an extension for the SUPERCOP framework developed with the objective of benchmarking hash functions on different microcontrollers in the context of the SHA-3 competition. XBX was the first project to measure, in a unified manner, the performance of cryptographic primitives built for different devices using the same evaluation methodology. In support for the now finished CAESAR competition, XBX was extended for AEAD algorithms and the ability to measure power consumption. However, apart from a 1-page summary of this so-called XXBX extension [6] (published in 2017), we are not of aware any further papers describing concrete details of its inner working, which indicates that XXBX is still under development.

Low-Level API. eBACS (and also XXBX) require AEAD implementations to follow a simple high-level API consisting of just two basic functions, namely `aead_encrypt` and `aead_decrypt`. This simplicity ensures that the API is easy to use (and hard to misuse), even for inexperienced software developers, but yields very coarse-grained results when applied to benchmarking. FELICS-AEAD, on the other hand, defines a low-level API comprising the seven functions specified in Listing 1. This low-level API offers a high degree of flexibility and allows for easy implementation of different kinds of security services, including the high-level functions of eBACS, for which nothing more than simple wrappers are needed. Consequently, adhering to the low-level API does not introduce more development effort than the high-level functions of eBACS. However, the low-level API enables a more fine-grained evaluation of AEAD algorithms since not only their overall execution times can be compared but also the times needed for initialization, encrypting/decrypting the data, processing the associated data, and generating/verifying the authentication tag. All these timings are valuable for algorithm designers when trying to analyze *why* a given AEAD algorithm is faster or slower than others. The fine-grained benchmarking results obtained with the low-level API may also be useful when one has to find the most suitable AEAD algorithm (out of a pool of candidates) for the encryption and/or authentication of a certain amount of data and associated data, respectively.

Evaluation Scenarios. eBACS measures the execution time of AEAD algorithms for combinations of data lengths and associated data lengths ranging from 0 to 2048 bytes in steps of one byte. These more than four million combinations have to be multiplied by the number of compiler options (i.e. optimization levels), which makes the collection of benchmarking results extremely computation-intensive and costly, especially when a large number of AEAD implementations have to be evaluated. The target platforms of eBACS (Intel/AMD and high-end ARM processors) are powerful enough to execute such a workload in an acceptable time, but this is not the case for resource-constrained 8 and 16-bit microcontrollers that can only be accessed via a debug probe and have to be programmed separately for each implementation. Using cycle-accurate instruction-set simulators is also not a solution since most of them lack a stable way of scripting to automate the verification of test vectors and the recording of cycle counts. These issues were the main reason to introduce the two evaluation scenarios (and six sub-scenarios) described in Subsect. 2.1. Namely, by defining very specific use cases that resemble real-world security services in the IoT, FELICS-AEAD becomes capable to evaluate a large number of implementations in a reasonable amount of time. The two scenarios are intended to have very different characteristics and requirements for AEAD algorithms. For example, the amount of data in Scenario 1 is relatively small and the length of the associated data is roughly a quarter of the data length. On the other hand, the amount of data in Scenario 2 is much higher, but the associated data amounts to only a small fraction of the data-length.

Figure-of-Merit. eBACS measures only the execution time of AEAD implementations, which makes it relatively easy to rank candidates by e.g. comparing their average throughput in cycles/byte. In contrast, FELICS-AEAD determines not only the execution time but also the memory footprint and code size of an implementation on each of the three supported platforms. This is reasonable since both RAM and ROM (resp. flash) are usually scarce resources in the IoT. However, taking three different metrics for each AEAD implementation into account makes a comparison of the benchmarking results relatively difficult, which is why FELICS allows the user to define a Figure-of-Merit (FOM) that combines execution time, RAM footprint, and code size into a single number. The FOM metric can use different weight factors for the three metrics, but by default, they have equal weight and, consequently, the execution time is considered to be equally important as RAM footprint and code size.

6 Conclusions and Final Remarks

In this paper, we introduced an extension to FELICS, a free and open-source benchmarking framework for the evaluation of AEAD algorithms. The main motivation behind this development is to give the designers of AEAD algorithms a fair, comprehensive and consistent way of evaluating their algorithms in the context of lightweight embedded devices, as well as a consistent way of comparing performance metrics between different algorithms. More specifically, this paper provided three contributions: (i) an API that allows a fine-grained evaluation of algorithms, while still maintaining design flexibility for the designers; (ii) a series of real-world based evaluations scenarios, allowing a fair comparison of algorithms based on their predicted future use; and (iii) preliminary results with a small set of well-known AEAD algorithms that demonstrate the framework's practical value. Thanks to its modular design, FELICS is very flexible and can be extended to support new metrics, new scenarios, and new devices. Furthermore, new implementations of AEAD algorithms can easily be added to the framework.

With that in mind, we encourage the cryptographic community to contribute optimized C and Assembler implementations of AEAD candidates submitted to the NIST lightweight crypto project and support in this way the fair and transparent evaluation of AEAD algorithms.

Acknowledgements. We would like to thank Daniel Dinu, Yann Le Corre, and Virat Shejwalkar for directly and indirectly helping with the development of this work. Luan Cardoso dos Santos is supported by the Luxembourg National Research Fund through grant PRIDE15/10621687/SPsquared.

References

1. Aumasson, J.-P., Jovanovic, P., Neves, S.: NORX: parallel and scalable AEAD. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8713, pp. 19–36. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11212-1_2
2. Beer, D.: MSPDebug: Debugging Tool for MSP430 MCUs (2015). http://mspdebug.sourceforge.net
3. Bellare, M., Rogaway, P.: Encode-then-encipher encryption: how to exploit nonces or redundancy in plaintexts for efficient cryptography. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 317–330. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_24
4. Bernstein, D.J., Lange, T.: eBACS: ECRYPT Benchmarking of Cryptographic Systems (2009). http://bench.cr.yp.to
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: CAESAR submission: Ketje v2 (2016)
6. Carter, M.R., Velagala, R.R., Pham, J., Kaps, J.P.: eXtended eXternal Benchmarking eXtension (XXBX). In: IEEE International Symposium on Hardware Oriented Security and Trust (HOST 2018) (2018)
7. CryptoLUX Team: FELICS: Fair Evaluation of Lightweight Cryptographic Systems (2016). http://www.cryptolux.org/index.php/FELICS
8. Dinu, D., Biryukov, A., Großschädl, J., Khovratovich, D., Corre, Y., Perrin, L.: FELICS-fair evaluation of lightweight cryptographic systems. In: NIST Workshop on Lightweight Cryptography, vol. 128 (2015)
9. Dinu, D., Le Corre, Y., Khovratovich, D., Perrin, L., Großschädl, J., Biryukov, A.: Triathlon of lightweight block ciphers for the Internet of Things. Cryptology ePrint Archive, Report 2015/209 (2015). https://eprint.iacr.org/2015/209
10. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1, submission to the CAESAR competition. CAESAR First Round Submission, March 2014
11. Katz, J., Yung, M.: Unforgeable encryption and chosen ciphertext secure modes of operation. In: Goos, G., Hartmanis, J., van Leeuwen, J., Schneier, B. (eds.) FSE 2000. LNCS, vol. 1978, pp. 284–299. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44706-7_20
12. McGrew, D., Viega, J.: The Galois/counter mode of operation (GCM). Submission to NIST Modes of Operation Process, vol. 20 (2004)
13. National Institute of Standards and Technology (NIST): Submission requirements and evaluation criteria for the lightweight cryptography standardization process. Technical report (2018). http://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf

14. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002), pp. 98–107. ACM Press, New York (2002)
15. Titzer, B.L., Lee, D.K., Palsberg, J.: Avrora: scalable sensor network simulation with precise timing. In: 2005 Fourth International Symposium on Information Processing in Sensor Networks (IPSN 2005), pp. 477–482. IEEE (2005)
16. Wenzel-Benner, C., Gräf, J.: XBX: eXternal Benchmarking eXtension for the SUPERCOP crypto benchmarking framework. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 294–305. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_20
17. Wu, H.: ACORN: a lightweight authenticated cipher (v3). Candidate for the CAESAR Competition (2016). https://competitions.cr.yp.to/round3/acornv3.pdf