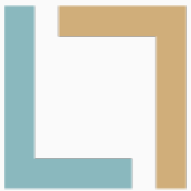
FELICS-AEAD: Benchmarking of Lightweight Authenticated Encryption Algorithms

Luan Cardoso dos Santos Johann Großschädl Alex Biryukov

University of Luxembourg - CryptoLux



Luxembourg National Research Fund

Supported by the Luxembourg National Research Fund through grant PRIDE15/10621687/SPsquared.

Overview

1. Introduction
2. The FELICS framework
   - Low-level API
   - Evaluation Scenarios
   - Platforms
   - Metrics
3. Preliminary Results
4. Comparison with other tools
5. Closing Remarks

Introduction

Introduction - AEAD

Definition

AE can be loosely defined as a symmetric cryptographic algorithm that is capable of, simultaneously, assuring the confidentiality *and* authenticity of data.

- AEAD was first formalized by Rogaway in 2002.
- Initially AEAD schemes were created by combining a block cipher in some mode of operation with a MAC algorithm.
- In recent years, many dedicated AEAD algorithms have been designed due to CAESAR and the NIST LWC project.

Motivation

- In response to NIST's call, 57 candidates were submitted by March 29, 2019 (CAESAR also had 57 submissions).
- In the course of CAESAR, the eBACS framework was used for benchmarking of the submitted algorithms. However, this tool mainly supports 64-bit Intel/AMD processors and relatively high-end ARM platforms.
- Many IoT devices are equipped with lower-end microcontrollers, e.g. 8-bit AVR, 16-bit MSP, and 32-bit ARM Cortex-M, which are optimized for small area and low power instead of high speed.

Features of FELICS AEAD

- Provide a low-level API that allows a more fine-grained measurement of the primitives, tailored specially for AEAD algorithms (high-level `encrypt` and `decrypt` functions are simply wrappers over the low-level functions).
- Provide a concise and fair way of measuring metrics of interest based on meaningful evaluation scenarios on different devices.

We provide both the API and Scenarios as an extension of the FELICS framework, available under GPLv3, aiming to improve transparency and reproducibility of benchmarking.

The FELICS framework

FELICS Framework

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.

FELICS Framework

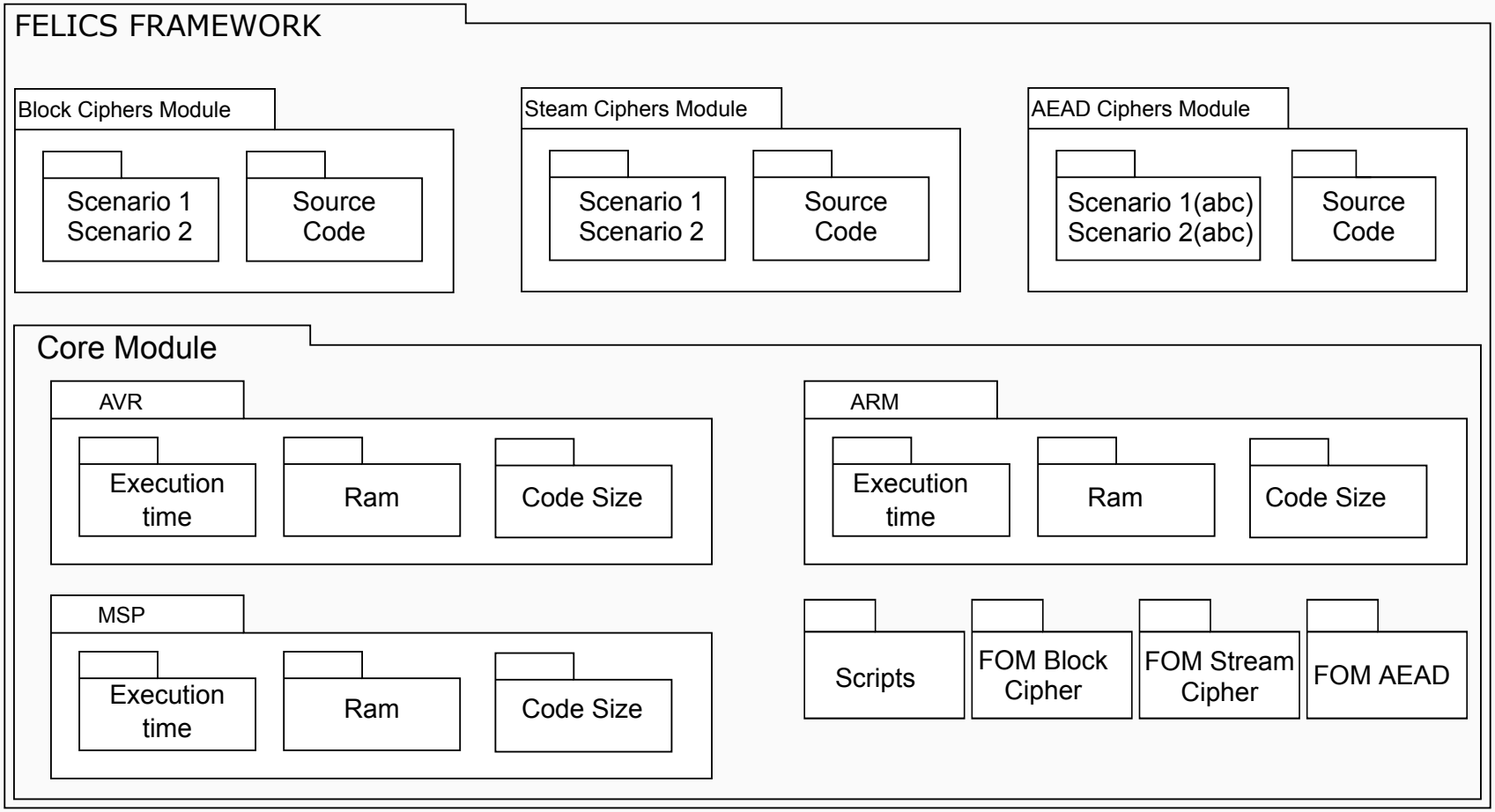


Figure 1: Modular structure of the FELICS benchmarking framework.

FELICS Framework

- The framework was designed to work on Linux and allows the benchmarking of C and ASM implementations.
- C is chosen for its popularity in IoT, and for being highly portable.
- Platform-specific Assembler, on the other hand, eliminates the impact of the compiler's ability for code optimization. Hand-crafted assembler code can take architecture-specific optimization into account, and usually outperforms C implementations.

The FELICS framework

Low-level API

Low-level API

- We aim to offer a generic and well-specified interface for the most common operations performed by an AEAD algorithm.
- Differently from other frameworks, FELICS API is composed of seven low-level functions instead of just `encrypt` and `decrypt`.
- Those functions give the framework more flexibility and allow the gathering of fine-grained benchmarking results.
- It is useful, for example, to better analyze why a given AEAD algorithm is faster or slower than it's competitors.

API

API function prototypes

```
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);
```

appendix

The FELICS framework

Evaluation Scenarios

Evaluation scenarios

- FELICS cipher evaluation is based on Scenarios.
- The scenarios implement real-world use cases, with practical relevance to IoT.
- With those scenarios, the main objective is generating realistic results that are meaningful in the real world, instead of a more artificial figure.
- The current scenarios are divided into three main groups:
  - Debug and Verification.
  - Security in IEEE 802.15.4 networks.
  - Security in IPv6 networks.

Test scenarios

Debug and Verification

Also referred to as “Scenario 0”, it is mainly used for debugging and testing implementations. The main objective is to help implementers check their code against know test vectors. It is also evaluates the low-level functions operating over a single block.

IEEE 802.15.4 – Scenario 1

Composed of three scenarios –Encryption, Authentication, and Encryption with authentication, it is based on the need for security in sensor networks and IoT applications.

IEEE 802.15.4 specifies a 127-byte frame, with (at most) 25 bytes of header and 107 bytes of payload.

Test scenarios

IPv6 – Scenario 2

Also composed of three scenarios in a similar manner to Scenario 1, these are based on the IPv6 frames, as defined in RFC 2460: at least 1280 bytes of payload and a fixed 40-byte header

appendix

The FELICS framework

Platforms

Platforms

- For this framework, three widely used microcontrollers were chosen to represent 8, 16, and 32-bit platforms:
  - AVR ATMega 128;
  - MSP430F1611;
  - Atmel SAM3X8 Cortex M3.
- These microcontrollers are optimized for small area and low power consumption instead of high performance.

Platforms

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

The FELICS framework

Metrics

Metrics

For cipher evaluation, the following metrics were used:

- **Execution time:** The measurement is extracted using a cycle-accurate simulator or a development board: `avrora`, `MSPDebug`, and ARM's performance registers.
- **RAM Consumption:** Static RAM consumption is measured using GNU `size` tool, while stack usage is measured via GDB.
- **Code Size:** Code size is measured with GNU `size` tool.

Figure of Merit

To aid in the classification of the evaluated ciphers, FELICS uses the *Figure-of-Merit*, that can be used to coalesce all the metrics into a single value. It uses a performance indicator, calculated, for each platform 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 the FoM defined as the average between AVR, MSP and ARM.

Preliminary Results

Benchmarking: Ideal-world vs real-world

General problem

The quality of benchmarking results depends on the quality of the implementations being benchmarked.

- Ideally, we want to quantify the efficiency of AEAD algorithms.
- In the real world, what we quantify is :
  - The skills and effort of the implementer;
  - The ability of the compiler to generate efficient code;
  - Hopefully also the efficiency of the algorithm.
- On the next slides, we show preliminary results for C implementations of five AEAD algorithms as a proof-of-concept for FELICS AEAD and *not necessarily* as evaluation of the AEAD algorithms.

Algorithms

For preliminary *proof-of-work*, we benchmarked the following algorithms:

- **ACORN** A stream-cipher-like CAESAR finalist created by Hongjun Wu.
- **AES-GCM** The Galois/Counter mode of operation over AES-128.
- **ASCON** A finalist of CAESAR, sponge-based algorithm created by Dobraunig et al.
- **Ketje** A third round CAESAR competitor, related to KECCAK, created by Bertoni et al.
- **NORX** A third round CAESAR competitor, sponge-based, and related to ChaCha's permutation, created by Aumasson et al.

Results – Scenario 1

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

Results – Scenario 2

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

Results – Observations i

- For Scenario 1, NORX is the clear winner, followed by ASCON and Ketje.
- For Scenario 2, NORX is still the winner, with more distance to ASCON and Ketje.
- The benchmarked version of NORX operates on 32-bit words, whereas ASCON and ACORN operates in 64-bit words. What seems a natural choice on high-end processors may lead to suboptimal performance on smaller microcontrollers.

Results – Observations ii

- C compilers for 8 and 16-bit platforms are not very good at handling 64-bit words.
- These small platforms also have a small register space, which might entail a high number of memory accesses.
- Finally, some “free” optimizations, such as ARM’s barrel shifter does not translate into 64-bit operands.

Comparison with other tools

Other tools i

- There are two important tools for benchmarking cryptography: eBACS and XXBX.
- eBACS was developed during the ECRYPT II project to evaluate the performance of crypto on Intel/AMD and Linux-capable high-end ARM processors (i.e. Cortex-A).
- eBACS features modules for public-key cryptosystems (eBATS), stream ciphers (eBASC), hash functions (eBASH), and AEAD (eBAEAD). Those modules operate under a common framework called SUPERCOP.

Other tools ii

- SUPERCOP – System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives– allows benchmarking of C, C++, and assembler implementations. Since execution time is the only metric measured by this framework, implementations are optimized for speed.
- The eXternal Benchmarking eXtension is an extension for SUPERCOP with the objective of measuring hash functions for the SHA-3 competition in different microcontrollers.

Comparison with FELICS i

- Low-Level API:
  - eBACS and XXBX require a high-level API consisting of `encrypt` and `decrypt` functions. While simple, this yields coarse-grained results.
  - FELICS uses a low-level API that can emulate eBACS API using wrapper functions and provides a low-level evaluation of AEAD algorithms. This can be useful for designers to better understand how the algorithm performs, as well as for users, to decide the best algorithm for their use-case.

Comparison with FELICS ii

- Evaluation Scenarios:
  - eBACS measures the execution time of AEAD algorithms for combinations of payload and AD lengths from 0 to 2048 bytes, which makes the collection of results computationally intensive.
  - While on the target architecture of eBACS it is not a big problem, resource-constrained 8 and 16-bit architectures pose a challenge, due to programming development boards, debug via probes, or simulators that are not friendly to scripting.
  - FELICS introduces the evaluation scenarios, that limit the number of data collection to acceptable levels, while still providing numbers relevant to real-world applications

Comparison with FELICS iii

- Metrics:
  - eBACS measures the time of AEAD implementations, which makes it relatively easy to rank candidates.
  - FELICS measures not only the execution time but also memory footprint and binary size, on each of the three supported platforms. This is reasonable since both RAM and ROM are scarce resources in IoT platforms.
  - To make comparisons easier, the FoM metric combines the values in an index, that can be tweaked to change the weight of each metric.

Closing Remarks

Closing Remarks

- We introduced FELICS AEAD, a free and open-source benchmarking framework for the evaluation of AEAD algorithms.
- Our main motivation is to give designers of AEAD algorithms a fair, comprehensive and consistent way of evaluating their algorithms in the context of lightweight embedded devices.
- We encourage the cryptographic community to contribute with optimized C and Assembler implementations of the candidates in the NIST lightweight crypto project.

Acknowledgments

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.

Thanks for your attention!

Questions?

API - Details

- **`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.

API - Details

- **`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 function 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.

back

Test scenarios: Details i

- **IEEE 802.15.4 Scenarios:** The maximum frame size of IEEE 802.15.4 is 127 bytes; the length of the header can not exceed 25 bytes. This leaves (at least) 102 bytes as frame payload.
  - **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.

Test scenarios: Details ii

- **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.
  - **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.

Back