FELICS – Fair Evaluation of Lightweight Cryptographic Systems

Daniel Dinu, Alex Biryukov, Johann Großschädl,
Dmitry Khovratovich, Yann Le Corre, Léo Perrin

SnT, University of Luxembourg

July 21, 2015

UNIVERSITÉ DU
LUXEMBOURG

Introduction

- **FELICS**
  - **F**air **E**valuation of **Li**ghtweight **C**ryptographic **S**ystems
  - open-source benchmarking framework for software implementations on constrained target devices widely used in the IoT

Introduction

- **FELICS**
  - **F**air **E**valuation of **Li**ghtweight **C**ryptographic **S**ystems
  - open-source benchmarking framework for software implementations on constrained target devices widely used in the IoT
- **Motivation**
  - lack of comparative performance figures

Introduction

- **FELICS**
  - **F**air **E**valuation of **Li**ghtweight **C**ryptographic **S**ystems
  - open-source benchmarking framework for software implementations on constrained target devices widely used in the IoT
- **Motivation**
  - lack of comparative performance figures
- **Outline**
  - this talk (FELICS): the framework structure and features
  - next talk (Triathlon): evaluation of 13 lightweight block ciphers using FELICS

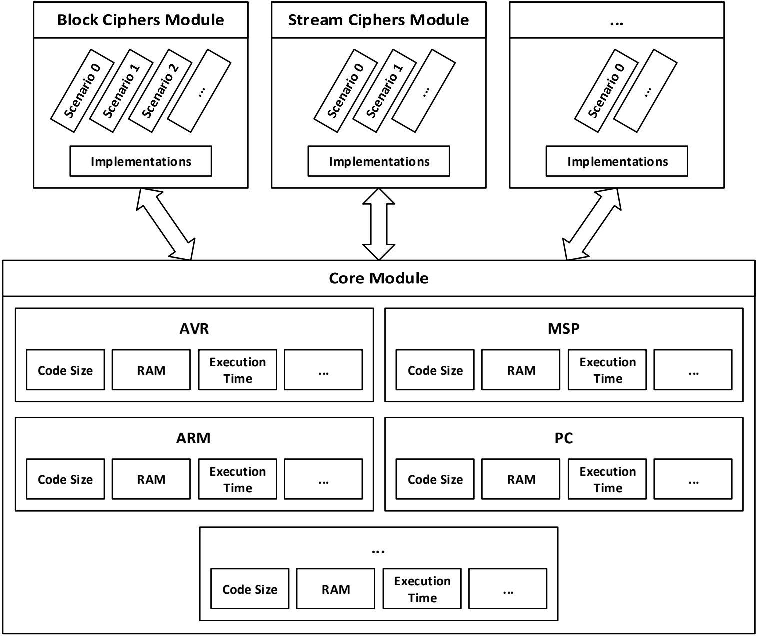
Related Work

	eBACS	ECRYPT II	BLOC	XBX	FELICS
Code Size	✗	✓	✓	✓	✓
RAM	✗	✓	✓	✓	✓
Exec. Time	✓	✓	✓	✓	✓
AVR	✗	✓	✗	✓	✓
MSP	✗	✗	✓	✓	✓
ARM	✗	✗	✗	✓	✓
PC	✓	✗	✗	✗	✓
Eval. Scen.	✗	✗	✗	✗	✓
Last Active	Nov '14	Nov '12	Jun '14	Nov '10	Jul '15

Goals

- **fair and consistent evaluation**
    - same assessment methodology for all implementations
- **accurate measurements and comprehensive results**
    - precise extraction of the metrics at operation level
- **free and open source**
    - widespread utilisation
- **flexible**
    - facilitates further development

Structure


Block Ciphers Module
Scenario 0
Scenario 1
Scenario 2
...
Implementations
Stream Ciphers Module
Scenario 0
Scenario 1
...
Implementations
...
Scenario 0
...
Implementations
Core Module
AVR
Code Size
RAM
Execution Time
...
MSP
Code Size
RAM
Execution Time
...
ARM
Code Size
RAM
Execution Time
...
PC
Code Size
RAM
Execution Time
...
...
Code Size
RAM
Execution Time
...

Core Module

- the heart of the framework
- provides the tools necessary to collect the metrics for each of the supported devices
- facilitates integration of new target devices and extracted metrics

Metric	AVR	MSP	ARM
Code Size	avr-size	msp430-size	arm-none-eabi-size
RAM	simavr avr-gdb	MSPDebug msp430-gdb	J-Link GDB Server arm-none-eabi-gdb
Execution Time	Avrora	MSPDebug	Arduino Due board

Block Ciphers Module

- same function signatures for all implementations
- template cipher implementation
- detailed implementation requirements
- implementation details in `implementation.info`

Function Signatures

```
void RunEncryptionKeySchedule(uint8_t *key, uint8_t *roundKeys);
void Encrypt(uint8_t *block, uint8_t *roundKeys);
void RunDecryptionKeySchedule(uint8_t *key, uint8_t *roundKeys);
void Decrypt(uint8_t *block, uint8_t *roundKeys);
```

Stream Ciphers Module

- same function signatures for all implementations
- template cipher implementation
- detailed implementation requirements
- implementation details in `implementation.info`

Function Signatures

```
void Setup(uint8_t *state, uint8_t *key, uint8_t *iv);
void Encrypt(uint8_t *state, uint8_t *stream, uint16_t length);
```

Target Devices

- Atmel AVR ATmega128
- Texas Instruments MSP430F1611
- Arduino Due board (ATSAM3X8E ARM Cortex-M3 MCU)

Characteristic	**AVR**	**MSP**	**ARM**
CPU	8-bit RISC	16-bit RISC	32-bit RISC
Frequency (MHz)	16	8	84
Registers	32	16	21
Architecture	Harvard	Von Neumann	Harvard
Flash (KB)	128	48	512
SRAM (KB)	4	10	96
EEPROM (KB)	4	-	-
Supply voltage (V)	4.5 - 5.5	1.8 - 3.6	1.6 - 3.6

Metrics

- three metrics:
  - code size (bytes)
  - RAM consumption (bytes)
  - execution time (cycles)
- accurate measurements
- detailed measurements ⇒ comprehensive results

Script

```
./collect_cipher_metrics.sh[{-h|--help}] [--version]
       [{-f|--format}=[0|1|2|3|4|5]]
       [{-a|--architectures}=['PC AVR MSP ARM']]
       [{-s|--scenarios}=['0 1 2']]
       [{-c|--ciphers}=['Cipher1 Cipher2 ...']]
```

Code Size

- the amount of information that is stored in the Flash memory of the target device
- the `GNU size` tool lists the section sizes and the total size in bytes for a given binary file

Code Size

- the amount of information that is stored in the Flash memory of the target device
- the GNU size tool lists the section sizes and the total size in bytes for a given binary file

Example

```
$ size operation.o
```

Code Size

- the amount of information that is stored in the Flash memory of the target device
- the GNU `size` tool lists the section sizes and the total size in bytes for a given binary file

Example

```
$ size operation.o
   text    data     bss     dec     hex filename
    266     128       0     394     18a operation.o
```

Code Size

- the amount of information that is stored in the Flash memory of the target device
- the GNU `size` tool lists the section sizes and the total size in bytes for a given binary file

Example

```
$ size operation.o
   text      data     bss     dec     hex filename
    266       128       0     394     18a operation.o
```

- `binary code size = size(text) + size(data)`

Code Size

- the amount of information that is stored in the Flash memory of the target device
- the GNU `size` tool lists the section sizes and the total size in bytes for a given binary file

Example

```
$ size operation.o
   text     data     bss     dec     hex filename
    266      128       0     394     18a operation.o
```

- `binary code size = size(text) + size(data)`
- `text` → code
- `data` → global initialized variables
- `bss` → global uninitialized variables

RAM

- RAM consumption is split into
  - data requirement (static RAM) → the size of the constants stored in target device RAM + scenario specific data
    - size of `data` section for object files
    - block size, key size, round keys size
  - stack requirement (dynamic RAM) → the maximum value of the RAM used to store local variables and return address after interrupts and subroutine calls

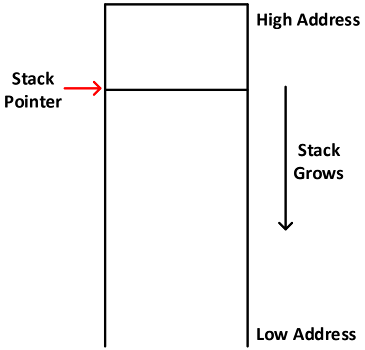
RAM

Stack

Code

```
    void BeginOperation()
→   {
        /* empty */
    }
    void Operation()
    {
        /* code */
    }
    void EndOperation()
    {
        /* empty */
    }
```

Stack



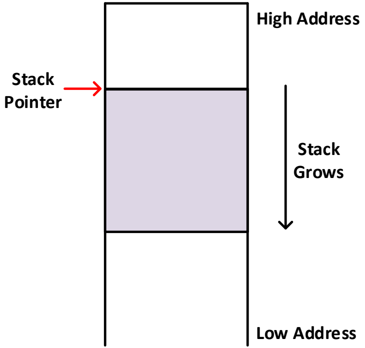
RAM

Stack

Code

```
    void BeginOperation()
→   {
        /* empty */
    }
    void Operation()
    {
        /* code */
    }
    void EndOperation()
    {
        /* empty */
    }
```

Stack



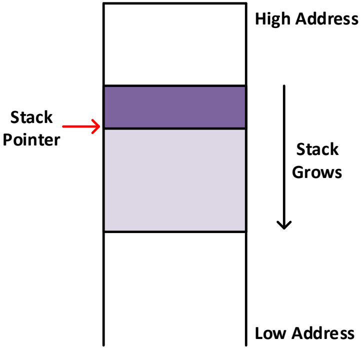
RAM

Stack

Code

```
    void BeginOperation()
    {
        /* empty */
    }
    void Operation()
    {
→       /* code */
    }
    void EndOperation()
    {
        /* empty */
    }
```

Stack



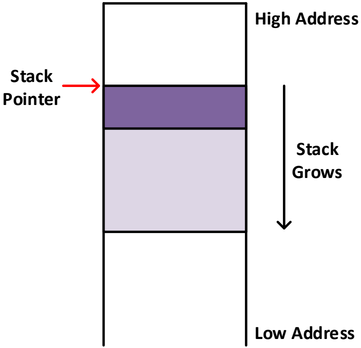
RAM

Stack

Code

```
    void BeginOperation()
    {
        /* empty */
    }
    void Operation()
    {
        /* code */
    }
    void EndOperation()
 →  {
        /* empty */
    }
```

Stack



Execution Time

- the number of CPU clock cycles spent on executing a given operation
- absolute difference between the system timer number of cycles at the end of the measured operation and at the beginning of the measured operation

Execution Time

- the number of CPU clock cycles spent on executing a given operation
- absolute difference between the system timer number of cycles at the end of the measured operation and at the beginning of the measured operation

Example

```
void Operation()
{
   /* code */
}
```

Execution Time

- the number of CPU clock cycles spent on executing a given operation
- absolute difference between the system timer number of cycles at the end of the measured operation and at the beginning of the measured operation

Example

```
    void Operation()
→   {
        /* code */
    }
```

$t_1 \leftarrow$ cycle count value

Execution Time

- the number of CPU clock cycles spent on executing a given operation
- absolute difference between the system timer number of cycles at the end of the measured operation and at the beginning of the measured operation

Example

```
     void Operation()
     {
        /* code */
→    }
```

$t_2 \leftarrow$ cycle count value

Execution Time

- the number of CPU clock cycles spent on executing a given operation
- absolute difference between the system timer number of cycles at the end of the measured operation and at the beginning of the measured operation

Example

```
void Operation()
{                          t1 ← cycle count value
   /* code */
}                          t2 ← cycle count value
```

- execution_time = $|t_2 - t_1|$

Results

Block Ciphers

- the time required to extract the metrics for 86 implementations of block ciphers in batch mode: 227 minutes
- the time required to extract each metric depends on many factors
- average values are computed for one run of each metric extraction process over all implementations

	AVR			MSP			ARM		
	Code Size [s]	RAM [s]	Exec. Time [s]	Code Size [s]	RAM [s]	Exec. Time [s]	Code Size [s]	RAM [s]	Exec. Time [s]
Scenario 0	0.85	3.78	1.54	1.05	10.85	1.06	1.38	15.53	16.40
Scenario 1	0.95	5.37	3.37	1.14	11.23	1.54	1.53	16.01	16.84
Scenario 2	0.97	3.61	1.68	1.13	8.22	1.11	1.54	13.54	15.82

Results

Stream Ciphers

- the time required to extract the metrics for 24 implementations of stream ciphers in batch mode: 30 minutes
- the time required to extract each metric depends on many factors
- average values are computed for one run of each metric extraction process over all implementations

	AVR			**MSP**			**ARM**		
	Code Size [s]	RAM [s]	Exec. Time [s]	Code Size [s]	RAM [s]	Exec. Time [s]	Code Size [s]	RAM [s]	Exec. Time [s]
Scenario 0	0.39	3.05	1.23	0.38	7.57	0.40	0.51	11.27	13.18
Scenario 1	0.39	3.11	1.31	0.37	7.57	0.40	0.50	11.25	13.17

Who can benefit?

- **designers of new ciphers**
  - understand how different components affect performance of the cipher
  - compare new algorithms with the state-of-the-art
- **software engineers**
  - select the best cipher to match the requirements of a particular application
- **standardization organizations**
  - conduct a fair and comprehensive evaluation of a large number of candidates

Conclusion

- designed & developed **FELICS**

Conclusion

- designed & developed **FELICS**
- fair and consistent evaluation of software implementations using the same target devices and measurement conditions

Conclusion

- designed & developed **FELICS**
- fair and consistent evaluation of software implementations using the same target devices and measurement conditions
- maintain the web page

https://www.cryptolux.org/index.php/FELICS

Conclusion

- designed & developed **FELICS**
- fair and consistent evaluation of software implementations using the same target devices and measurement conditions
- maintain the web page

  https://www.cryptolux.org/index.php/FELICS

  - source code
  - comprehensive results
  - FOM scripts
  - VM with all necessary tools pre-installed

Conclusion

- designed & developed **FELICS**
- fair and consistent evaluation of software implementations using the same target devices and measurement conditions
- maintain the web page

  https://www.cryptolux.org/index.php/FELICS
  - source code
  - comprehensive results
  - FOM scripts
  - VM with all necessary tools pre-installed
- encourage the community to contribute with implementations

Conclusion

- designed & developed **FELICS**
- fair and consistent evaluation of software implementations using the same target devices and measurement conditions
- maintain the web page

  https://www.cryptolux.org/index.php/FELICS
  - source code
  - comprehensive results
  - FOM scripts
  - VM with all necessary tools pre-installed
- encourage the community to contribute with implementations

Future Work

- new modules (e.g. authenticated encryption, . . .)
- new metrics (e.g. power consumption, . . .)
- new target devices
- new evaluation scenarios
- contributions to the framework are welcome

Future Work

- new modules (e.g. authenticated encryption, . . .)
- new metrics (e.g. power consumption, . . .)
- new target devices
- new evaluation scenarios
- contributions to the framework are welcome

Thank You!

https://www.cryptolux.org/index.php/FELICS