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
FELICS

F E A R  E v a l u a t i o n  o f  L i g h t w e i g h t  C r y p t o g r a p h i c  S y s t e m s

open-source benchmarking framework for software implementations on constrained target devices widely used in the IoT
Introduction

- **FELICS**
  - Fair Evaluation of Lightweight Cryptographic Systems
  - open-source benchmarking framework for software implementations on constrained target devices widely used in the IoT

- **Motivation**
  - lack of comparative performance figures
Introduction

- **FELICS**
  - Fair Evaluation of Lightweight Cryptographic Systems
  - 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

<table>
<thead>
<tr>
<th></th>
<th>eBACS</th>
<th>ECRYPT II</th>
<th>BLOC</th>
<th>XBX</th>
<th>FELICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RAM</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Exec. Time</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AVR</td>
<td>×</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSP</td>
<td>×</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ARM</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PC</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Eval. Scen.</td>
<td>×</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Last Active</td>
<td>Nov ’14</td>
<td>Nov ’12</td>
<td>Jun ’14</td>
<td>Nov ’10</td>
<td>Jul ’15</td>
</tr>
</tbody>
</table>
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
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

<table>
<thead>
<tr>
<th>Metric</th>
<th>AVR</th>
<th>MSP</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>avr-size</td>
<td>msp430-size</td>
<td>arm-none-eabi-size</td>
</tr>
<tr>
<td>RAM</td>
<td>simavr</td>
<td>MSPDebug</td>
<td>J-Link GDB Server</td>
</tr>
<tr>
<td></td>
<td>avr-gdb</td>
<td>msp430-gdb</td>
<td>arm-none-eabi-gdb</td>
</tr>
<tr>
<td>Execution Time</td>
<td>Avrora</td>
<td>MSPDebug</td>
<td>Arduino Due board</td>
</tr>
</tbody>
</table>
Block Ciphers Module

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

Function Signatures

```c
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

```c
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)

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

<table>
<thead>
<tr>
<th>text</th>
<th>data</th>
<th>bss</th>
<th>dec</th>
<th>hex</th>
<th>filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>266</td>
<td>128</td>
<td>0</td>
<td>394</td>
<td>18a</td>
<td>operation.o</td>
</tr>
</tbody>
</table>
```

- binary code size = \( \text{size(text)} + \text{size(data)} \)
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

<table>
<thead>
<tr>
<th>text</th>
<th>data</th>
<th>bss</th>
<th>dec</th>
<th>hex</th>
<th>filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>266</td>
<td>128</td>
<td>0</td>
<td>394</td>
<td>18a</td>
<td>operation.o</td>
</tr>
</tbody>
</table>

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

text → code

data → global initialized variables

bss → global uninitialized variables
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
void BeginOperation()
{
    /* empty */
}

void Operation()
{
    /* code */
}

void EndOperation()
{
    /* empty */
}
RAM
Stack

Code

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

Stack
RAM

Stack

Code

```c
void BeginOperation()
{
    /* empty */
}

void Operation()
{
    /* code */
}

void EndOperation()
{
    /* empty */
}
```

Stack

- High Address
- Low Address
- Stack grows
- Stack Pointer
### RAM

**Stack**

#### Code

```c
void BeginOperation()
{
    /* empty */
}

void Operation()
{
    /* code */
}

void EndOperation()
{
    /* empty */
}
```

#### Stack Diagram

- **High Address**
- **Low Address**
- **Stack Grows**
- **Stack Pointer**
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

```c
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

```c
void Operation()
{
    t1 ← cycle count value
    /* 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

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

    t2 ← cycle count value
```

(Umiversity of Luxembourg)
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

```c
void Operation()
{
    t_1 ← cycle count value
    /* code */
    t_2 ← 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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 0</td>
<td>0.85</td>
<td>3.78</td>
<td>1.54</td>
<td>1.05</td>
<td>10.85</td>
<td>1.06</td>
<td>1.38</td>
<td>15.53</td>
<td>16.40</td>
</tr>
<tr>
<td>Scenario 1</td>
<td>0.95</td>
<td>5.37</td>
<td>3.37</td>
<td>1.14</td>
<td>11.23</td>
<td>1.54</td>
<td>1.53</td>
<td>16.01</td>
<td>16.84</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>0.97</td>
<td>3.61</td>
<td>1.68</td>
<td>1.13</td>
<td>8.22</td>
<td>1.11</td>
<td>1.54</td>
<td>13.54</td>
<td>15.82</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>AVR</th>
<th>MSP</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[s]</td>
<td>[s]</td>
<td>[s]</td>
</tr>
<tr>
<td><strong>Scenario 0</strong></td>
<td>0.39</td>
<td>3.05</td>
<td>1.23</td>
</tr>
<tr>
<td><strong>Scenario 1</strong></td>
<td>0.39</td>
<td>3.11</td>
<td>1.31</td>
</tr>
</tbody>
</table>
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