Lightweight Cryptography on ARM

Software implementation of block ciphers and ECC

Hayato Fujii, Rafael Cruz, Tiago Reis
Diego F. Aranha, Julio López, Harsh Kupwade Patil

University of Campinas, LG Electronics Inc.
Introduction
Cryptography can mitigate critical security issues in embedded devices.

<table>
<thead>
<tr>
<th>Security property</th>
<th>Technique</th>
<th>Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protecting data at rest</td>
<td>FS-level encryption</td>
<td>Block cipher</td>
</tr>
<tr>
<td>Protecting data in transit</td>
<td>Secure channel</td>
<td>Auth block/stream cipher</td>
</tr>
<tr>
<td>Secure software updates</td>
<td>Code signing</td>
<td>Digital signatures</td>
</tr>
<tr>
<td>Secure booting</td>
<td>Integrity/Authentication</td>
<td>Hash functions, MACs</td>
</tr>
<tr>
<td>Secure debugging</td>
<td>Entity authentication</td>
<td>Challenge-response</td>
</tr>
<tr>
<td>Device id/auth</td>
<td>Auth protocol</td>
<td>PKC</td>
</tr>
<tr>
<td>Key distribution</td>
<td>Key exchange</td>
<td>PKC</td>
</tr>
</tbody>
</table>

Several algorithms required to implement primitives:

- Block and stream ciphers
- Hash functions
- AEAD and Message Authentication Codes (MACs)
- Elliptic Curve Cryptography
Problem: Why “lightweight cryptography”? Shouldn’t all cryptography be ideally lightweight?

From Mouha in [Mou15]

“Although the question seems simple, this appears to be a quite controversial subject. (...) It is important to note that lightweight cryptography should not be equated with weak cryptography”.

Solution: Alternative name for application-specific cryptography or application-driven cryptographic design?
We discuss techniques for efficient and secure implementations of lightweight encryption in software:

1. FANTOMAS, an LS-Design proposed in [GLSV14].
2. PRESENT, a Substitution-Permutation Network (SPN) [BKL+07].
3. CURVE25519 for Elliptic Curve Cryptography.

We target low-end and NEON-capable ARM processors, typical of embedded systems. Results are part of a project sponsored by LG involving 7 students and more than 30 symmetric (C) and asymmetric (ASM) algorithms.
Fantomas
LS-Designs

Paradigm to construct block ciphers providing:

- **Lightweight** designs from simple substitution and linear layers.
- Friendliness to *side-channel countermeasures* (*bitslicing* and *masking*).
- Tweakable variant for **authenticated encryption** (*SCREAMv3*).
Algorithm 1 LS-Design encrypting block $B$ into ciphertext $C$ with key $K$.

1: $C \leftarrow B \oplus K$ \hfill $\triangleright$ $C$ represents an $s \times l$-bit matrix
2: $\textbf{for } 0 \leq r < N_r \textbf{ do}$
3: \hspace{1em} $\textbf{for } 0 \leq i < l \textbf{ do}$ \hfill $\triangleright$ S-box layer
4: \hspace{2em} $C[i, \star] = S[C[i, \star]]$
5: \hspace{1em} $\textbf{end for}$
6: \hspace{1em} $\textbf{for } 0 \leq j < s \textbf{ do}$ \hfill $\triangleright$ L-box layer
7: \hspace{2em} $C[\star, j] = L[C[\star, j]]$
8: \hspace{1em} $\textbf{end for}$
9: $C \leftarrow C \oplus K \oplus C(r)$ \hfill $\triangleright$ Key and round constant addition
10: $\textbf{end for}$
11: $\textbf{return } C$
The LS-Design paper introduced an involutive instance (Robin), and a non-involutive cipher (Fantomas).

**Fantomas**

- **128-bit** key length and block size.
- **No** key scheduling.
- 8-bit (3/5-bit 3-round) **S-boxes** from MISTY.
- **L-box** from vector-matrix product in $\mathbb{F}_2$. 
Implementation in 32/64 bits

Internal state can be represented with union to respect strict aliasing rules for 16/32/64-bit operations:

```c
typedef union {
    uint32_t u32;     // uint64_t u64;
    uint16_t u16[2];  // uint16_t u16[4];
} U32_t;
```

Bitsliced S-boxes operate over 16-bit chunks in the u16 portion.

**Key addition** works using the u32/u64 internal state:

```c
for (j = 0; j < 4; j++)   // for (j = 0; j < 2; j++)
    st[j].u32 ^= key_32[j]; // st[j].u64 ^= key_64[j];
```
Implementation in 32/64 bits

L-box can be evaluated using **two precomputed tables**:

```c
/* Unprotected L-box version */
st[j].u16[0] = LBoxH[st[j].u16[0] >> 8] ^
               LBoxL[st[j].u16[0] & 0xff];

               LBoxL[st[j].u16[1] & 0xff];
```

**Problem:** Beware of **cache-timing attacks**!
L-box can be evaluated using **two precomputed tables**: 

```c
/* Unprotected L-box version */
st[j].u16[0] = LBoxH[st[j].u16[0]>>8] ^ LBoxL[st[j].u16[0] & 0xff];
```

```c
```

**Problem:** Beware of **cache-timing attacks**!

Attacker who monitors **L-box positions in cache** can recover internal state. Internal state trivially reveals **keys and plaintext** if recovered right before/after last/first key addition.
Algorithm 2 LS-Design encrypting block $B$ into ciphertext $C$ with key $K$.

1. $C \leftarrow B \oplus K$  \hspace{1cm} \triangleright C$ represents an $s \times l$-bit matrix
2. for $0 \leq r < N_r$ do
3. \hspace{1cm} for $0 \leq i < l$ do  \hspace{1cm} \triangleright S-box layer
4. \hspace{2cm} $C[i, \star] = S[C[i, \star]]$
5. \hspace{2cm} end for
6. \hspace{1cm} for $0 \leq j < s$ do  \hspace{1cm} \triangleright L-box layer
7. \hspace{2cm} $C[\star, j] = L[C[\star, j]]$
8. \hspace{2cm} end for
9. \hspace{1cm} $C \leftarrow C \oplus K \oplus C(r)$ \hspace{1cm} \triangleright Key and round constant addition
10. end for
11. return $C$
Solution: We can replace memory access with online computation:

```c
static inline type_t LBox(type_t x, type_t y, uint8_t s) {
    x &= y;
    x ^= x >> 8;
    x ^= x >> 4;
    x ^= x >> 2;
    x ^= x >> 1;
    return (x & 0x00010001) << s;
    // return (x & 0x0001000100010001) << s
}
```
L-boxes can be evaluated using **shuffling** instructions to compute **8 table lookups** in parallel.

**Important:** 32-bit implementations can process 2 blocks and vector implementations can process **16 blocks** simultaneously in CTR mode.
Counter transformation for the vectorized CTR implementation:

(a) Initial state of the counter

(b) Final state of the counter
Key must be transformed to follow representation.
Experiments I

Benchmark: Encrypt+decrypt 128 bytes in CBC or encrypt 128 bits in CTR mode.

- Related work: FELICS (triathlon of block ciphers) [DCK+15].
- Platforms:
  1. Cortex-M3 (Arduino Due, 32 bits):
     - GCC 4.8.4 from Arduino with flags 
       `-O3 -fno-schedule-insns -mcpu=cortex-m3 -mthumb`.  
     - Cycles count by converting the output of the `micros()` function.
  2. Cortex-M4 (Teensy 3, 32 bits):
     - GCC 4.8.4 from Arduino with flags 
       `-O3 -fno-schedule-insns -mcpu=cortex-m3 -mthumb`.  
     - Cycles counts through CCNT register.
  3. Cortex-A53 (ODROID OC2, 64 bits):
     - GCC 6.1.1 with flags 
       `-O3 -fno-schedule-insns -mcpu=cortex-a53 -mthumb -march=native`.  
     - Cycles counts through CCNT register.
Results

Fantomas in CBC mode

Arduino Due Cortex-M3

Cycle Count

<table>
<thead>
<tr>
<th></th>
<th>Ours</th>
<th>FELICS Fast</th>
<th>FELICS Compact</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32-bit CT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Code Size (ROM)

<table>
<thead>
<tr>
<th></th>
<th>Ours</th>
<th>FELICS Fast</th>
<th>FELICS Compact</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32-bit CT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results

Fantomas in CTR mode

Arduino Due Cortex-M3

Cycle Count

Code Size (ROM)

Implementation

32-bit

32-bit CT

Ours
FELICS Fast
FELICS Compact

32-bit 32-bit CT
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Results

Fantomas in CBC mode

Cortex-M3/M4/A53

Cycles Per Byte (CPB)

32-bit CBC | 32-bit CT CBC | 64-bit CBC | 64-bit CT CBC
0 | 500 | 1000 | 1500 | 2000 | 2500

Cortext-M3 (Ours) | Cortex-M4 (Ours) | Cortex-A53 (Ours)

Implementation

Code Size (ROM)

32-bit CBC | 32-bit CT CBC | 64-bit CBC | 64-bit CT CBC
0 | 1000 | 2000 | 3000 | 4000 | 5000 | 6000

Cortext-M3 (Ours) | Cortex-M4 (Ours) | Cortex-A53 (Ours)
Benchmark: Encrypt 128 bits in CTR mode.

- **Related work**: Adjusted timings from SCREAMv3 presentation in the CAESAR competition [GLS+15].

- **Platforms**:
  1. **Cortex-A15 (ODROID XU4, 32 bits + NEON)**:
     - GCC 6.1.1 with flags `-O3 -fno-schedule-insn
      -mcpu=cortex-a15 -mthumb -march=native.
     - Cycles count through CCNT register.
  2. **Cortex-A53 (ODROID OC2, 64 bits + NEON)**:
     - GCC 6.1.1 with flags `-O3 -fno-schedule-insn
      -mcpu=cortex-a53 -mthumb -march=native.
     - Cycles counts through CCNT register.
Results

Fantomas in CTR mode

NEON implementation

Cycles Per Byte (CPB)

Cortex-A15  Cortex-A53

Fantomas (Ours)
16-block version (Ours)
16-block version (RW)

Platform

Code Size (ROM)

Cortex-A15  Cortex-A53

0 1000 2000 3000 4000 5000 6000 7000 8000

20
1. **Constant time implementation** against cache-timing attacks:
   - Performance penalty of 3 times in low-end ARMs.
   - *Inherent* in vector implementations.
   - Not sufficient against *other* side-channel attacks.

2. **Masked implementation** against power attacks:
   - *Significant* quadratic performance penalty (almost twice slower with a single mask).
   - Not sufficient against *cache timing attacks*.
   - *Key masking* to force attacker to recover all shares (additional 10-20% overhead).
Fantomas has some limitations regarding side-channel resistance:

- S-boxes do not require tables, but are expensive to mask.
- L-boxes are free to mask, but expensive to compute in constant time.

New state-of-the-art implementations of Fantomas:

- Portable implementation in C is 35% and 52% faster than [DCK+15] on Cortex-M, and similar in code size.
- New countermeasures against cache timing attacks.
- NEON implementation is 40% faster in ARM than [GLS+15].
PRESENT
Proposed in 2007 and standardized by ISO/IEC, one of the first lightweight block cipher designs.

**PRESENT**

- Substitution-permutation network.
- 80-bit or 128-bit key and 64-bit block.
- Key schedule for 31 rounds with 64-bit subkeys \( \text{subkey}_i \).
- 4-bit S-boxes with Boolean representation friendly to **bitslicing**.
- Bit permutation \( P \) such that \( P^2 = P^{-1} \).
Algorithm

Figure 2: 4-bit S-Boxes in PRESENT.

<table>
<thead>
<tr>
<th>$x$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S[x]$</td>
<td>c</td>
<td>5</td>
<td>6</td>
<td>b</td>
<td>9</td>
<td>0</td>
<td>a</td>
<td>d</td>
<td>3</td>
<td>e</td>
<td>f</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

$$P(i) = \begin{cases} 16i \mod 63 & \text{if } i \neq 63 \\ 63 & \text{if } i = 63 \end{cases}$$
Algorithm 3 PRESENT encrypting block $B$ to ciphertext block $C$.

1. $C \leftarrow B$
2. for $i = 1$ to $31$ do
3. $\quad C \leftarrow C \oplus \text{subkey}_i$
4. $\quad C \leftarrow S(C)$
5. $\quad C \leftarrow P(C)$
6. end for
7. $C \leftarrow P \oplus \text{subkey}_{32}$
8. return $C$
### Implementation

#### PRESENT optimizations

1. Decompose permutation $P^2$ in **software-friendly** involutive permutations $P_0$ and $P_1$.
2. **Rearrange** rounds to accommodate new permutations.
3. **Efficient** bitsliced S-boxes from [CHM11].
4. For CTR mode in 32 bits, process two blocks simultaneously.
Implementation

Figure 3: Permutation $P$ in PRESENT.

$$A = \begin{bmatrix}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\
48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 & 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \\
\end{bmatrix}$$

$$P(A) = \begin{bmatrix}
0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 & 32 & 36 & 40 & 44 & 48 & 52 & 56 & 60 \\
1 & 5 & 9 & 13 & 17 & 21 & 25 & 29 & 33 & 37 & 41 & 45 & 49 & 53 & 57 & 61 \\
2 & 6 & 10 & 14 & 18 & 22 & 26 & 30 & 34 & 38 & 42 & 46 & 50 & 54 & 58 & 62 \\
\end{bmatrix}$$

Figure 4: Permutations $P_0$ and $P_1$ for optimized PRESENT.

$$P_0(A) = \begin{bmatrix}
0 & 16 & 32 & 48 & 4 & 20 & 36 & 52 & 8 & 24 & 40 & 56 & 12 & 28 & 44 & 60 \\
1 & 17 & 33 & 49 & 5 & 21 & 37 & 53 & 9 & 25 & 41 & 57 & 13 & 29 & 45 & 61 \\
2 & 18 & 34 & 50 & 6 & 22 & 38 & 54 & 10 & 26 & 42 & 58 & 14 & 30 & 46 & 62 \\
\end{bmatrix}$$

$$P_1(A) = \begin{bmatrix}
0 & 1 & 2 & 3 & 16 & 17 & 18 & 19 & 32 & 33 & 34 & 35 & 48 & 49 & 50 & 51 \\
4 & 5 & 6 & 7 & 20 & 21 & 22 & 23 & 36 & 37 & 38 & 39 & 52 & 53 & 54 & 55 \\
8 & 9 & 10 & 11 & 24 & 25 & 26 & 27 & 40 & 41 & 42 & 43 & 56 & 57 & 58 & 59 \\
\end{bmatrix}$$
Algorithm 4 PRESENT encrypting block $B$ to ciphertext block $C$.

1: $C \leftarrow B$
2: for $i = 1$ to $15$ do
3: $C \leftarrow C \oplus \text{subkey}_{2i-1}$
4: $C \leftarrow P_0(C)$
5: $C \leftarrow S(C)$
6: $C \leftarrow P_1(C)$
7: $C \leftarrow C \oplus P(\text{subkey}_{2i})$
8: $C \leftarrow S(C)$
9: end for
10: $C \leftarrow P \oplus \text{subkey}_{31}$
11: $C \leftarrow P(C)$
12: $C \leftarrow S(C)$
13: $C \leftarrow C \oplus \text{subkey}_{32}$
14: return $C$
Experiments I

**Benchmark:** Encrypt+decrypt+key schedule 128 bytes in CBC or encrypt 128 bits in CTR mode.

- **Related work:** ASM implementation in FELICS [DCK+15], 2nd-order constant-time masked ASM implementation of PRESENT [dGPdLP+16].

- **Platforms:**
  1. **Cortex-M3 (Arduino Due, 32 bits):**
     - GCC 4.8.4 from Arduino with flags `-O3 -fno-schedule-insns -mcpu=cortex-m3 -mthumb`.
     - Cycles count by converting the output of the `micros()` function.
  2. **Cortex-M4 (Teensy 3.2, 32 bits):**
     - GCC 4.8.4 from Arduino with flags `-O3 -fno-schedule-insns -mcpu=cortex-m3 -mthumb`.
     - Cycles counts through CCNT register.
Results

PRESENT

E+D+KS 128 bytes (CBC) or encrypt 128 bits (CTR) on ARM Cortex-M3

Cycle Count

32-bit CBC 32-bit CTR

0 50000 100000 150000 200000 250000 300000

Ours FELICS

Implementation

Code Size (ROM)

32-bit CBC 32-bit CTR

0 500 1000 1500 2000 2500 3000

31
Results

PRESENT

E+D+KS 128 bytes (CBC) or encrypt 128 bits (CTR) on ARM Cortex-M4

- Constant time (Ours)
- Masked (RW)

**Cycle Count**

- 32-bit CBC: 60000
- 32-bit CTR: 30000

**Code Size (ROM)**

- 32-bit CBC: 1600
- 32-bit CTR: 1200
Conclusions

Side-channel resistance:

- PRESENT can be efficiently implemented in constant time.
- Performance penalty from masking is lower than Fantomas, mainly due to choice of S-boxes.

New state-of-the-art implementations of PRESENT:

- S-boxes can be bitsliced (no tables) and permutations can be made much faster.
- Performance improvement of 8x factor.
- Our constant-time CTR implementation is now among the fastest block ciphers in the FELICS benchmark (competitive with SPARX).
Detailed timings

Table 1: Comparison of block ciphers implemented in C by this work with AES in Assembly for encrypting 128 bits in CTR mode across long messages.

<table>
<thead>
<tr>
<th>Block cipher</th>
<th>Cortex-M3 Unprotected</th>
<th>Cortex-M3 CT</th>
<th>Cortex-M4 Unprotected</th>
<th>Cortex-M4 CT</th>
<th>Cortex-M4 ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fantomas</td>
<td>2291</td>
<td>9063</td>
<td>2191</td>
<td>7866</td>
<td>1272</td>
</tr>
<tr>
<td>PRESENT-80</td>
<td>-</td>
<td>2052</td>
<td>-</td>
<td>1597</td>
<td>1124</td>
</tr>
<tr>
<td>AES-128 [SS16]</td>
<td>546</td>
<td>1617</td>
<td>554</td>
<td>1618</td>
<td>12120</td>
</tr>
</tbody>
</table>
Curve25519
Difficult choice of multiplication instructions in Cortex-M3 \([\text{dG15}]\):

- **MUL**: effectively \(16 \times 16 \rightarrow 32\), 1 cycle.
- **MLA (acc)**: effectively \(16 \times 16 \rightarrow 32\), 2 cycles.
- **UMULL**: \(32 \times 32 \rightarrow 64\), 3-5 cycles.
- **UMLAL**: \(32 \times 32 \rightarrow 64\), 4-7 cycles.

Side-channel attack known using early-terminating multiplications for ECDH \([\text{GOPT09}]\), although not clear if applicable to laddering. Countermeasures replace UMULL with instructions costing 12-19 cycles \([\text{Ham11}]\).

**Important**: At this penalty, Cortex-M0 implementation \([\text{DHH}^{+15}]\) should still be competitive.
Previous work in constant time with Karatsuba over reduced radix \([dG15]\).

Alternative implementation on Cortex-M4:

- Full-radix to enjoy arithmetic density and single-cycle multiplications.
- Comba with register allocation inspired by operand caching \([HW11]\).
- Arithmetic closely follow ideas from the full-radix Cortex-M0 implementation.
- Check next presentation. :)
Table 2: Experimental results for different implementations of randomized X25519 and Ed25519 on ARM processors. The figures include timings for the field arithmetic and protocol operations. Measurements for latency in clock cycles were taken as the average of 1000 executions by benchmarking code directly in the M4 board.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Ours</th>
<th>De Santis et al.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>86 cc</td>
<td>106 cc</td>
</tr>
<tr>
<td>Subtraction</td>
<td>86 cc</td>
<td>108 cc</td>
</tr>
<tr>
<td>Multiplication</td>
<td>276 cc</td>
<td>546 cc</td>
</tr>
<tr>
<td>Squaring</td>
<td>252 cc</td>
<td>362 cc</td>
</tr>
<tr>
<td>Inversion</td>
<td>66,634 cc</td>
<td>96,337 cc</td>
</tr>
<tr>
<td>X25519</td>
<td>907,240 cc</td>
<td>1,563,582 cc</td>
</tr>
<tr>
<td>Code size of X25519</td>
<td>4,152B of ROM</td>
<td>3,786B of ROM</td>
</tr>
<tr>
<td>Ed25519 Signature</td>
<td>813,300 cc</td>
<td>-</td>
</tr>
<tr>
<td>Code size for Signature</td>
<td>22,268B of ROM</td>
<td>-</td>
</tr>
<tr>
<td>Ed25519 Verification</td>
<td>1,265,078 cc</td>
<td>-</td>
</tr>
<tr>
<td>Code size for Signature</td>
<td>28,240B of ROM</td>
<td>-</td>
</tr>
</tbody>
</table>

**Important:** figures updated in June 2017 to include contributions due to Hayato Fujii.
Important: All timings cross-checked with the MPS2 ARM development board provided by LG.

Fantomas for x86/SSE can be found at https://github.com/rafajunio/fantomas-x86.
Questions?
**PRESENT: an ultra-lightweight block cipher.**

N. Courtois, D. Hulme, and T. Mourouzis. 
**Solving circuit optimisation problems in cryptography and cryptanalysis.**

D. Dinu, Y. L. Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov. 
**Triathlon of lightweight block ciphers for the internet of things.**
W. de Groot.  
*A performance study of X25519 on Cortex M3 and M4*, 2015.

**Bitsliced masking and arm: Friends or foes?**  

**High-speed curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers.**  

CAESAR candidate SCREAM Side-Channel Resistant Authenticated Encryption with Masking.  

LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations.  
J. Großschädl, E. Oswald, D. Page, and M. Tunstall.  
**Side-channel analysis of cryptographic software via early-terminating multiplications.**  

F. B. Hamouda.  
**Exploration of efficiency and side-channel security of different implementations of rsa.**  
2011.

M. Hutter and E. Wenger.  
**Fast multi-precision multiplication for public-key cryptography on embedded microprocessors.**  
N. Mouha.  
The design space of lightweight cryptography.  

P. Schwabe and K. Stoffelen.  
All the AES You Need on Cortex-M3 and M4.  