Lightweight Cryptography on ARM

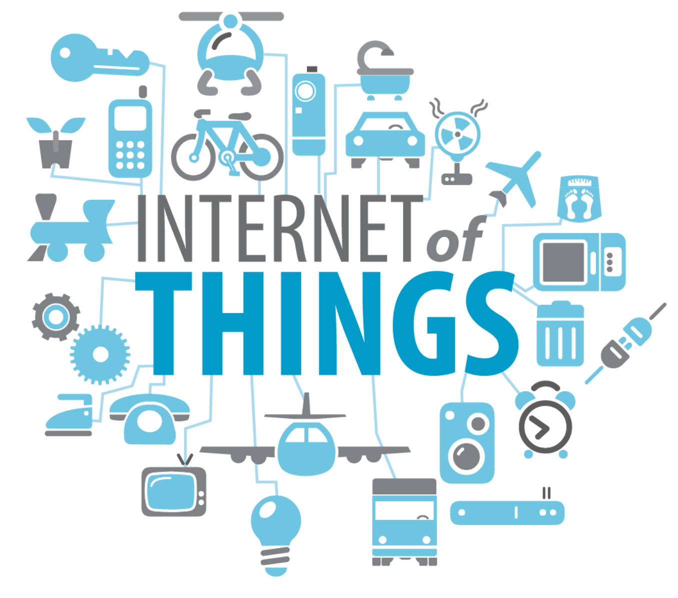
Software implementation of block ciphers and ECC

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

University of Campinas, LG Electronics Inc.

Introduction

Context



Context

Cryptography can mitigate critical security issues in embedded devices.

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

Several algorithms required to implement primitives:

- Block and stream ciphers
- Hash functions
- AEAD and Message Authentication Codes (MACs)
- Elliptic Curve Cryptography

Context

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*?

Summary

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.

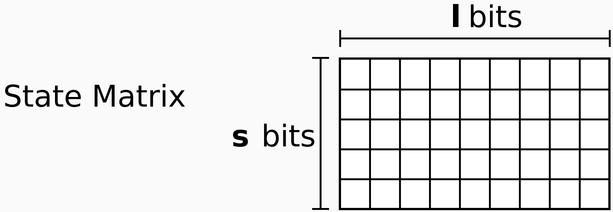
Fantomas

Construction

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).



Construction

Algorithm 1 LS-Design encrypting block B into ciphertext C with key K.

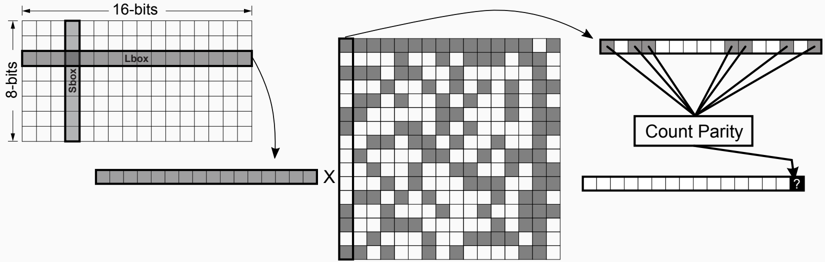
$C \leftarrow B \oplus K$ ▷ C represents an $s \times l$-bit matrix
for $0 \leq r < N_r$ **do**
    for $0 \leq i < l$ **do** ▷ S-box layer
        $C[i, \star] = S[C[i, \star]]$
    end for
    for $0 \leq j < s$ **do** ▷ L-box layer
        $C[\star, j] = L[C[\star, j]]$
    end for
    $C \leftarrow C \oplus K \oplus C(r)$ ▷ Key and round constant addition
end for
return C

Algorithm

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:

```
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:

```
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**:

```
/* Unprotected L-box version */
st[j].u16[0] = LBoxH[st[j].u16[0]>>8] ^
               LBoxL[st[j].u16[0] & 0xff];
st[j].u16[1] = LBoxH[st[j].u16[1]>>8] ^
               LBoxL[st[j].u16[1] & 0xff];
```

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

Implementation in 32/64 bits

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

```
/* Unprotected L-box version */
st[j].u16[0] = LBoxH[st[j].u16[0]>>8] ^
               LBoxL[st[j].u16[0] & 0xff];
st[j].u16[1] = LBoxH[st[j].u16[1]>>8] ^
               LBoxL[st[j].u16[1] & 0xff];
```

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.

Construction

Algorithm 2 LS-Design encrypting block B into ciphertext C with key K.

$C \leftarrow B \oplus K$ ▷ C represents an $s \times l$-bit matrix
for $0 \leq r < N_r$ **do**
  for $0 \leq i < l$ **do** ▷ S-box layer
    $C[i, \star] = S[C[i, \star]]$
  end for
  for $0 \leq j < s$ **do** ▷ L-box layer
    $C[\star, j] = L[C[\star, j]]$
  end for
  $C \leftarrow C \oplus K \oplus C(r)$ ▷ Key and round constant addition
end for
return C

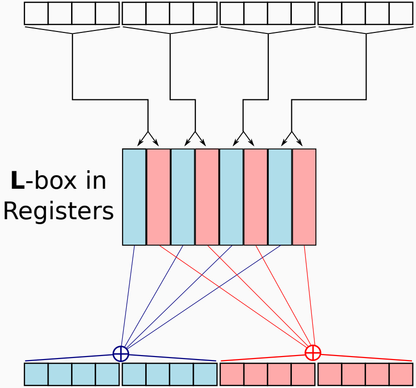
Implementation in 32/64 bits

Solution: We can replace memory access with online computation:

```
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
}
```

NEON implementation

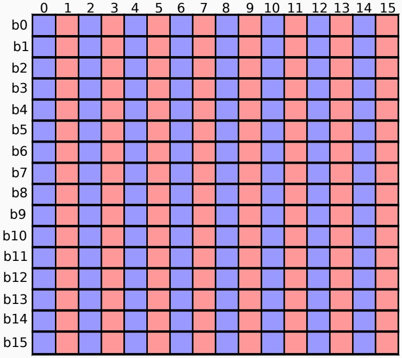
L-boxes can be evaluated using **shuffling** instructions to compute **8 table lookups** in parallel.



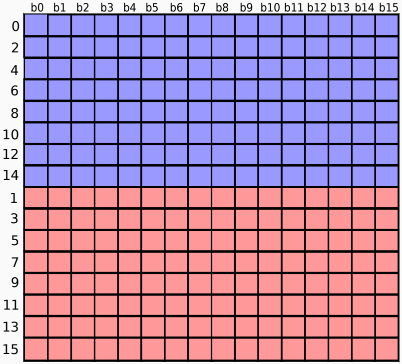
Important: 32-bit implementations can process 2 blocks and vector implementations can process **16 blocks** simultaneously in CTR mode.

NEON implementation

Counter transformation for the vectorized CTR implementation:



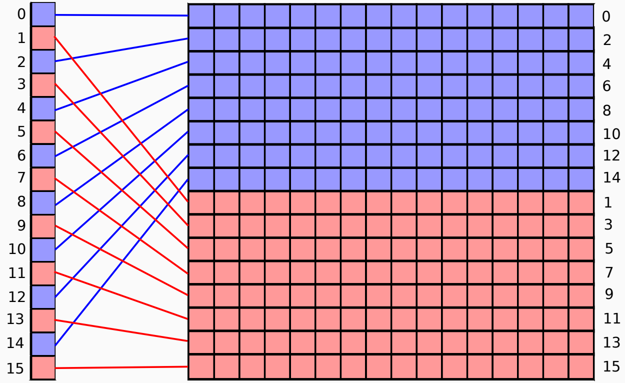
(a) Initial state of the counter



(b) Final state of the counter

NEON implementation

Key must be transformed to follow representation.

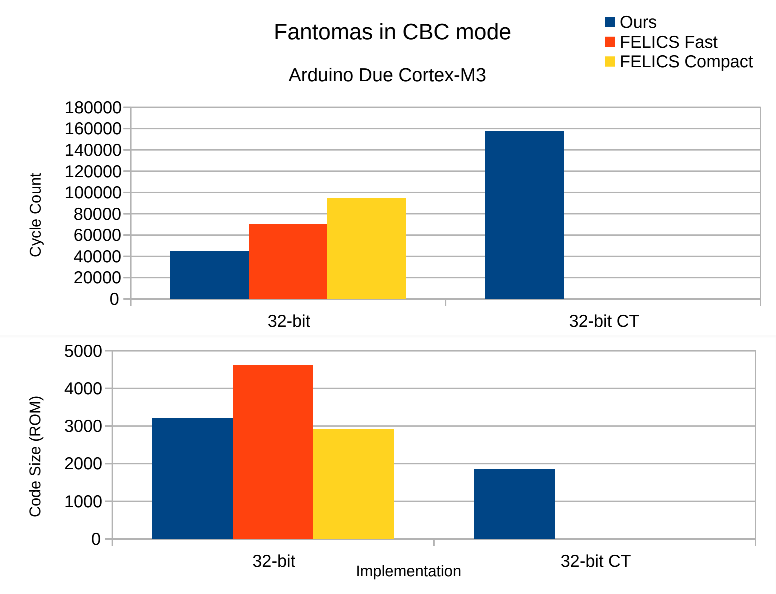


Experiments I

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

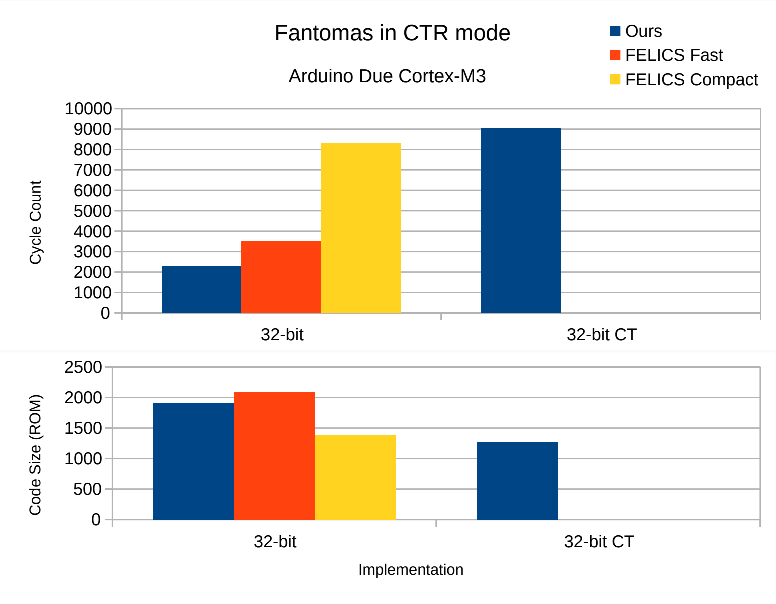
- **Related work:** FELICS (triathlon of block ciphers) [DCK+15].
- **Platforms:**
  1. **Cortex-M3 (Arduino Due, 32 bits):**
     - GCC 4.8.4 from Arduino with flags `-O3 -fno-schedule-insns -mcpu=cortex-m3 -mthumb`.
     - Cycles count by converting the output of the `micros()` function.
  2. **Cortex-M4 (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



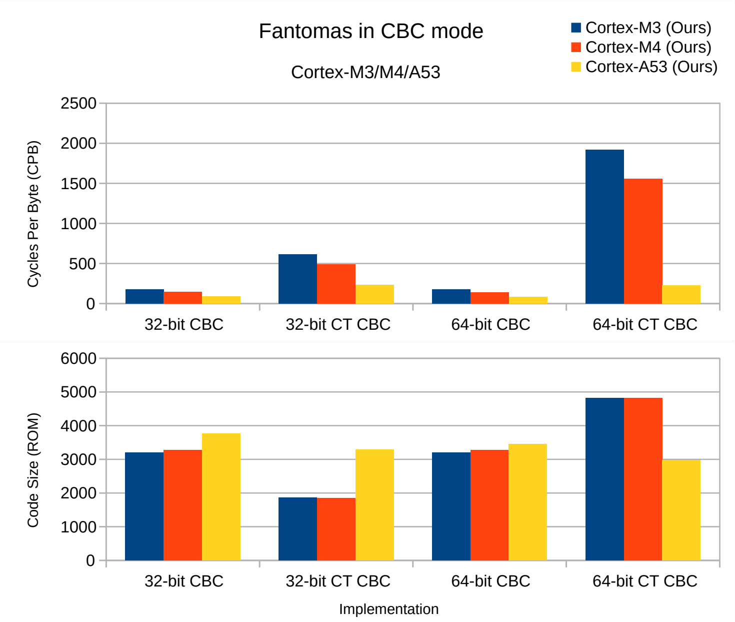
Fantomas in CBC mode
Arduino Due Cortex-M3
Ours
FELICS Fast
FELICS Compact
Cycle Count
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
32-bit
32-bit CT
Code Size (ROM)
5000
4000
3000
2000
1000
0
32-bit
32-bit CT
Implementation

Results



Fantomas in CTR mode
Arduino Due Cortex-M3
Ours
FELICS Fast
FELICS Compact
Cycle Count
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
32-bit
32-bit CT
Code Size (ROM)
2500
2000
1500
1000
500
0
32-bit
32-bit CT
Implementation

Results

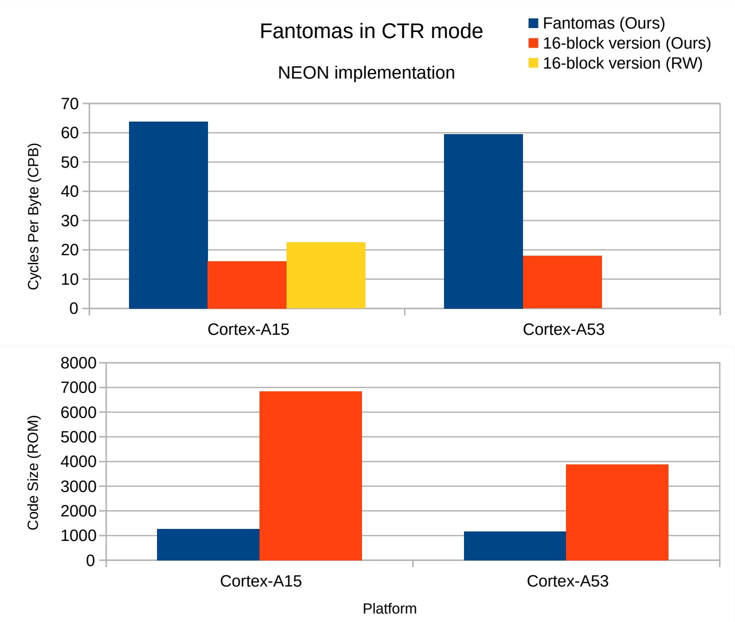

Fantomas in CBC mode
Cortex-M3/M4/A53
Cortex-M3 (Ours)
Cortex-M4 (Ours)
Cortex-A53 (Ours)
Cycles Per Byte (CPB)
2500
2000
1500
1000
500
0
32-bit CBC
32-bit CT CBC
64-bit CBC
64-bit CT CBC
Code Size (ROM)
6000
5000
4000
3000
2000
1000
0
32-bit CBC
32-bit CT CBC
64-bit CBC
64-bit CT CBC
Implementation

Experiments II

Benchmark: Encrypt 128 bits in CTR mode.

- **Related work**: Ajusted 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-insns -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-insns -mcpu=cortex-a53 -mthumb -march=native`.
     - Cycles counts through CCNT register.

Results



Fantomas in CTR mode
NEON implementation
Fantomas (Ours)
16-block version (Ours)
16-block version (RW)
Cycles Per Byte (CPB)
70
60
50
40
30
20
10
0
Cortex-A15
Cortex-A53
Code Size (ROM)
8000
7000
6000
5000
4000
3000
2000
1000
0
Cortex-A15
Cortex-A53
Platform

Side-channel resistance

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).

Conclusions

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

Algorithm

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 $subkey_i$.
- 4-bit S-boxes with Boolean representation friendly to **bitslicing**.
- Bit permutation P such that $P^2 = P^{-1}$.

Algorithm

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2

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

$$P(i) = \begin{cases} 16i \mod 63 & \text{if } i \neq 63 \\ 63 & \text{if } i = 63 \end{cases}$$

Algorithm

Algorithm 3 PRESENT encrypting block B to ciphertext block C.

$C \leftarrow B$
for $i = 1$ **to** 31 **do**
    $C \leftarrow C \oplus subkey_i$
    $C \leftarrow S(C)$
    $C \leftarrow P(C)$
end for
$C \leftarrow P \oplus subkey_{32}$
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

$$A = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 & 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix}$$

$$P(A) = \begin{bmatrix} 0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 & 32 & 36 & 40 & 44 & 48 & 52 & 56 & 60 \\ 1 & 5 & 9 & 13 & 17 & 21 & 25 & 29 & 33 & 37 & 41 & 45 & 49 & 53 & 57 & 61 \\ 2 & 6 & 10 & 14 & 18 & 22 & 26 & 30 & 34 & 38 & 42 & 46 & 50 & 54 & 58 & 62 \\ 3 & 7 & 11 & 15 & 19 & 23 & 27 & 31 & 35 & 39 & 43 & 47 & 51 & 55 & 59 & 63 \end{bmatrix}$$

Figure 3: Permutation P in PRESENT.

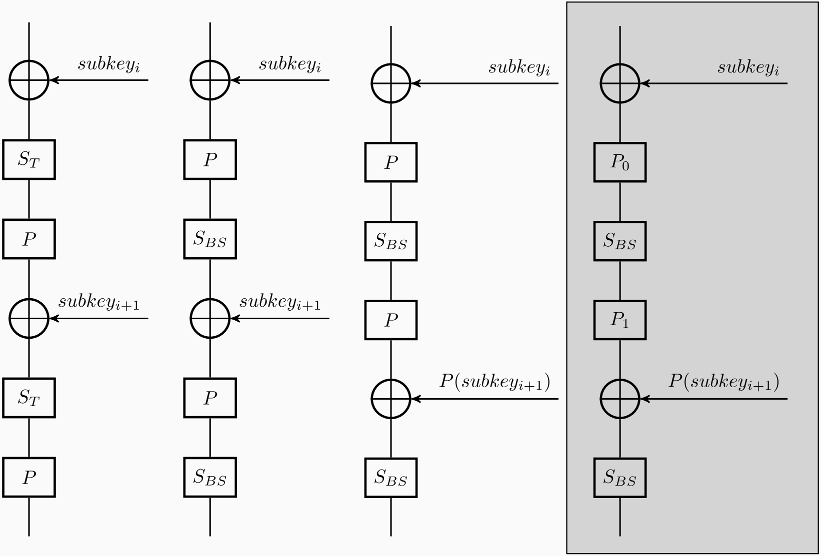
$$P_0(A) = \begin{bmatrix} 0 & 16 & 32 & 48 & 4 & 20 & 36 & 52 & 8 & 24 & 40 & 56 & 12 & 28 & 44 & 60 \\ 1 & 17 & 33 & 49 & 5 & 21 & 37 & 53 & 9 & 25 & 41 & 57 & 13 & 29 & 45 & 61 \\ 2 & 18 & 34 & 50 & 6 & 22 & 38 & 54 & 10 & 26 & 42 & 58 & 14 & 30 & 46 & 62 \\ 3 & 19 & 35 & 51 & 7 & 23 & 39 & 55 & 11 & 27 & 43 & 59 & 15 & 31 & 47 & 63 \end{bmatrix}$$

$$P_1(A) = \begin{bmatrix} 0 & 1 & 2 & 3 & 16 & 17 & 18 & 19 & 32 & 33 & 34 & 35 & 48 & 49 & 50 & 51 \\ 4 & 5 & 6 & 7 & 20 & 21 & 22 & 23 & 36 & 37 & 38 & 39 & 52 & 53 & 54 & 55 \\ 8 & 9 & 10 & 11 & 24 & 25 & 26 & 27 & 40 & 41 & 42 & 43 & 56 & 57 & 58 & 59 \\ 12 & 13 & 14 & 15 & 28 & 29 & 30 & 31 & 44 & 45 & 46 & 47 & 60 & 61 & 62 & 63 \end{bmatrix}$$

Figure 4: Permutations P_0 and P_1 for optimized PRESENT.

Implementation

f



Implementation

Algorithm 4 PRESENT encrypting block B to ciphertext block C.

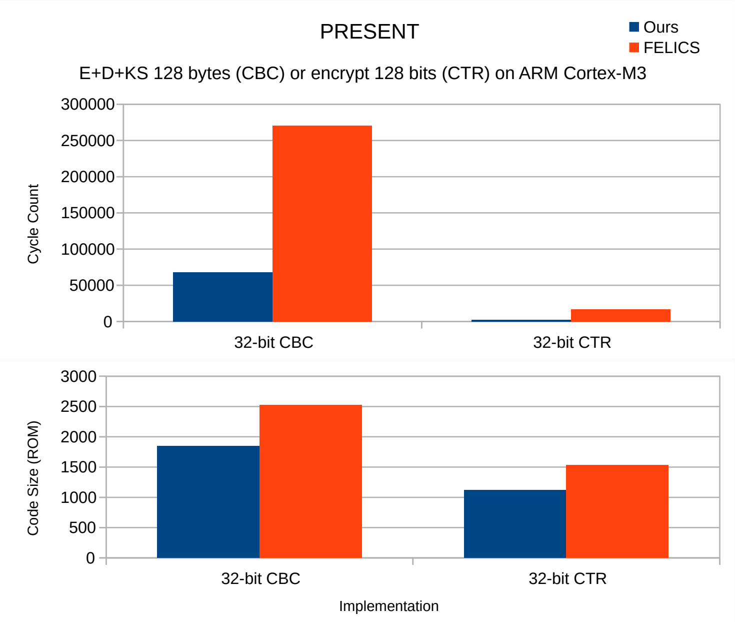
$C \leftarrow B$
for $i = 1$ **to** 15 **do**
  $C \leftarrow C \oplus subkey_{2i-1}$
  $C \leftarrow P_0(C)$
  $C \leftarrow S(C)$
  $C \leftarrow P_1(C)$
  $C \leftarrow C \oplus P(subkey_{2i})$
  $C \leftarrow S(C)$
end for
$C \leftarrow P \oplus subkey_{31}$
$C \leftarrow P(C)$
$C \leftarrow S(C)$
$C \leftarrow C \oplus subkey_{32}$
return C

Experiments I

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

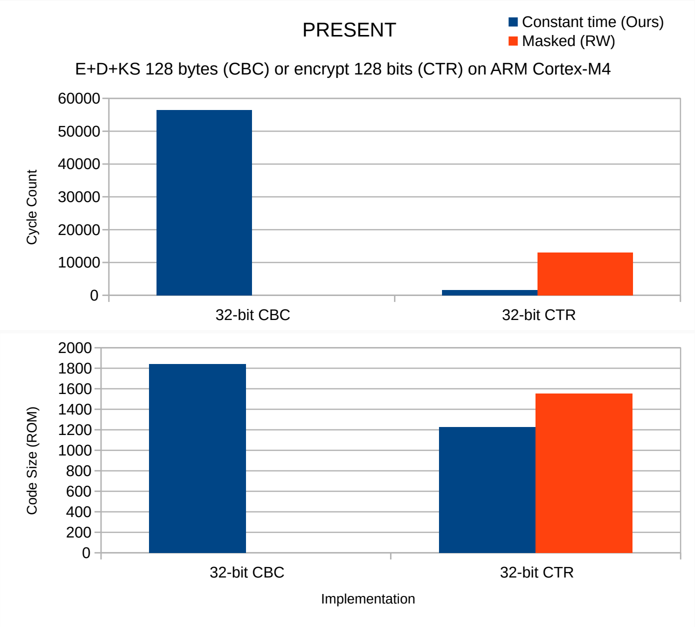
- **Related work:** ASM implementation in FELICS [DCK+15], 2nd-order constant-time masked ASM implementation of PRESENT [dGPdLP+16].
- **Platforms:**
  1. **Cortex-M3 (Arduino Due, 32 bits):**
     - GCC 4.8.4 from Arduino with flags `-O3 -fno-schedule-insns -mcpu=cortex-m3 -mthumb`.
     - Cycles count 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
Ours
FELICS
E+D+KS 128 bytes (CBC) or encrypt 128 bits (CTR) on ARM Cortex-M3
Cycle Count
300000
250000
200000
150000
100000
50000
0
32-bit CBC
32-bit CTR
Code Size (ROM)
3000
2500
2000
1500
1000
500
0
32-bit CBC
32-bit CTR
Implementation

Results


PRESENT
Constant time (Ours)
Masked (RW)
E+D+KS 128 bytes (CBC) or encrypt 128 bits (CTR) on ARM Cortex-M4
Cycle Count
60000
50000
40000
30000
20000
10000
0
32-bit CBC
32-bit CTR
Code Size (ROM)
2000
1800
1600
1400
1200
1000
800
600
400
200
0
32-bit CBC
32-bit CTR
Implementation

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.

	Cortex-M3		Cortex-M4		
Block cipher	**Unprotected**	**CT**	**Unprotected**	**CT**	**ROM**
Fantomas	2291	9063	2191	7866	1272
PRESENT-80	-	2052	-	**1597**	**1124**
AES-128 [SS16]	546	1617	554	1618	12120

Curve25519

Field arithmetic in $\mathbb{F}_{2^{255}-19}$

Difficult choice of multiplication instructions in Cortex-M3 [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 [GOPT09], although not clear if applicable to laddering. Countermeasures replace UMULL with instructions costing 12-19 cycles [Ham11].

Important: At this penalty, Cortex-M0 implementation [DHH$^+$15] should still be competitive.

Field arithmetic in $\mathbb{F}_{2^{255}-19}$

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. :)

Result

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.

Operation	**Ours**	**Next presentation :)**
Addition	**85** cc	106 cc
Subtraction	**85** cc	108 cc
Multiplication	**532** cc	546 cc
Squaring	532 cc	**362** cc
Inversion	140,306 cc	**96,337** cc
X25519	**1,607,860** cc	1,658,083 cc
Code size of X25519	3,102B of ROM	**2,952**B of ROM
Signature	1,122,709 cc	-
Verification	2,747,329 cc	-
Code size for Ed25519	32,210B of ROM	-

Final notes

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?

References I

A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe.
PRESENT: an ultra-lightweight block cipher.
In *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

N. Courtois, D. Hulme, and T. Mourouzis.
Solving circuit optimisation problems in cryptography and cryptanalysis.
IACR Cryptology ePrint Archive, 2011:475, 2011.

D. Dinu, Y. L. Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov.
Triathlon of lightweight block ciphers for the internet of things.
IACR Cryptology ePrint Archive, 2015:209, 2015.

References II

- W. de Groot.
  A performance study of X25519 on Cortex M3 and M4, 2015.
- W. de Groot, K. Papagiannopoulos, A. de La Piedra, E. Schneider, and L. Batina.
  Bitsliced masking and arm: Friends or foes?
  Cryptology ePrint Archive, Report 2016/946, 2016.
  `http://eprint.iacr.org/2016/946`.
- M. Düll, B. Haase, G. Hinterwälder, M. Hutter, C. Paar, A. H. Sánchez, and P. Schwabe.
  High-speed curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers.
  Des. Codes Cryptography, 77(2-3):493–514, 2015.

V. Grosso, G. Laurent, F. Standaert, K. Varici, F. Durvaux, L. Gaspar, and S. Kerckhof.
CAESAR candidate SCREAM Side-Channel Resistant Authenticated Encryption with Masking.
http://2014.diac.cr.yp.to/slides/leurent-scream.pdf, 2015.

V. Grosso, G. Leurent, F. Standaert, and K. Varici.
LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations.
In *FSE*, volume 8540 of *Lecture Notes in Computer Science*, pages 18–37. Springer, 2014.

References IV

J. Großschädl, E. Oswald, D. Page, and M. Tunstall.
Side-channel analysis of cryptographic software via early-terminating multiplications.
In *ICISC*, volume 5984 of *Lecture Notes in Computer Science*, pages 176–192. Springer, 2009.

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.
In *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 459–474. Springer, 2011.

N. Mouha.
The design space of lightweight cryptography.
Cryptology ePrint Archive, Report 2015/303, 2015.
`http://eprint.iacr.org/2015/303`.

P. Schwabe and K. Stoffelen.
All the AES You Need on Cortex-M3 and M4.
Cryptology ePrint Archive, Report 2016/714, 2016.
`http://eprint.iacr.org/2016/714`.