

# Fast Implementation of Elliptic Curve Cryptography and Pairing Computation for Sensor Networks

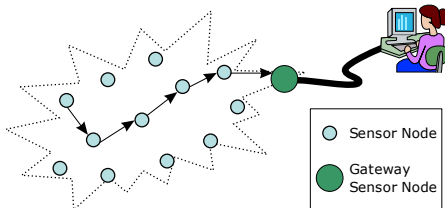
Julio López  
University of Campinas  
Campinas, Brazil

Joint work with  
Diego Aranha, Danilo Câmara, Ricardo Dahab, Leonardo Oliveira  
and Conrado Lopes

The 13th Workshop on Elliptic Curve Cryptography (ECC-2009),  
Calgary, Canada

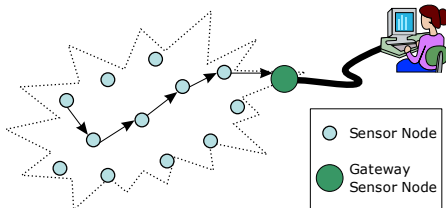
- Motivation: ECC and Pairings for WSNs
- Field arithmetic in  $\mathbb{F}_{2^m}$  and  $\mathbb{F}_p$
- Elliptic curve cryptography for WSNs
  - Binary curves for ATmega128L
  - Prime curves for MSP430
- Pairing computation for WSNs
  - The  $\eta_T$  pairing for  $E/\mathbb{F}_{2^{271}}$  ATmega128L and ARMv5
  - Pairings at the 128-bit security level for MSP430

# Wireless Sensor Networks



- A WSN is an ad hoc network comprised of tiny and smart sensing devices employed for cooperative monitoring tasks.

# Wireless Sensor Networks



- A WSN is an ad hoc network comprised of tiny and smart sensing devices employed for cooperative monitoring tasks.
- The deployment of cryptography on sensor networks is a challenging task, given the limited computational power and the resource-constrained nature of the sensing devices.
- A typical node has an 8-bit processor at 8 MHz with less than 128 KB of instruction memory and 4KB of RAM memory.

# Challenge: security services for WSNs

- In the context of WSNs, the issue of securing and authenticating communications is a difficult one since the nodes have no capacity for the secure storage of secret keys.

# Challenge: security services for WSNs

- In the context of WSNs, the issue of securing and authenticating communications is a difficult one since the nodes have no capacity for the secure storage of secret keys.
- Application of PKC to WSNs:
  - Elliptic Curve Cryptography;
  - Pairing-Based Cryptography.

# Challenge: security services for WSNs

- In the context of WSNs, the issue of securing and authenticating communications is a difficult one since the nodes have no capacity for the secure storage of secret keys.
- Application of PKC to WSNs:
  - Elliptic Curve Cryptography;
  - Pairing-Based Cryptography.
- The purpose of this talk is to illustrate the performance of software implementation for pairing computation and point multiplication on [three](#) WSN processors.

## Contributions:

- Optimization techniques for arithmetic in  $\mathbb{F}_{2^m}$  and  $\mathbb{F}_p$ ;
- Fast software implementation of ECC; ECDSA
- Efficient software implementation of pairing computations.

- ATmega128L: 8-bit processor (MICAz Mote)  
ECC-163, ECC-233,  $\eta_T$  pairing in  $\mathbb{F}_{2^{271}}$
- MSP430: 16-bit processor (Tmote Sky, TelosB)  
ECC-160, ECC-256, BN curves at 128-bit level of security
- Intel PXA27x: 32-bit processor (Imote2)  
 $\eta_T$  pairing in  $\mathbb{F}_{2^{271}}$  (at 80-bit level of security)



Figure: MICAz mote

- ATmega128L, low-power CMOS 8 bit microcontroller, 7.3828 MHz of clock frequency;
- 4KB of RAM memory, 128KB of ROM memory;
- Simple two-stage pipeline;
- Limited shift instructions;
- Memory instructions (addressing, reads, writes) cost 2 cycles

ATMega128 is a typical RISC processor:

- 32 registers, but 6 of them are special for pointers;
- 1 register for memory/arithmetic temporary values;
- Only  $32 - 6 - 1 = 25$  useful registers.

Relevant instructions:

<b>Instruction</b>	<b>Description</b>	<b>Cost</b>
rsl, lsl	Right/left shift by 1-bit	1 cycle
swap	Swap high and low nibbles	1 cycle
bld, bst	Bit load/store from/to flag	1 cycle
eor	Exclusive bitwise OR	1 cycle
ld, st	Memory load/store	2 cycles
adiw, sbiw	Pointer arithmetic	2 cycles

- Polynomial basis:  $a(z) \in \mathbb{F}_{2^m} = \sum_{i=0}^{m-1} a_i z^i$   
(field elements are binary polynomials of degree  $< m$ )
- Irreducible polynomial:  $f(z)$  (trinomial or pentanomial)
- Software representation: vector of  $n = \lceil m/w \rceil$  words  
( $w = 8, 16, 32$ )
- Graphical representation:

**A** **A**<sub>n-1</sub> ... **A**<sub>9</sub> **A**<sub>8</sub> **A**<sub>7</sub> **A**<sub>6</sub> **A**<sub>5</sub> **A**<sub>4</sub> **A**<sub>3</sub> **A**<sub>2</sub> **A**<sub>1</sub> **A**<sub>0</sub>

- Binary fields:  $\mathbb{F}_{2^m}$ ,  $m = 163, 233, 271, 283$
- Arithmetic operations:
  - Addition (simple logical XOR)
  - Squaring
  - Modular reduction
  - Square-root
  - Multiplication
  - Inversion

Irreducible polynomials:  $f(z)$ 

- NIST irreducibles polynomials:
  - $\mathbb{F}_{2^{163}}$ :  $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$
  - $\mathbb{F}_{2^{233}}$ :  $f(z) = z^{233} + z^{74} + 1$
  - $\mathbb{F}_{2^{283}}$ :  $f(z) = z^{283} + z^{12} + z^7 + z^5 + 1$
- Square-root friendly irreducible polynomials [Hankerson, Avanzi, Scott]
  - $\mathbb{F}_{2^{163}}$ :  $f(z) = z^{163} + z^{57} + z^{49} + z^{29} + 1$
  - $\mathbb{F}_{2^{283}}$ :  $f(z) = z^{283} + z^{97} + z^{89} + z^{87} + 1$
  - $\mathbb{F}_{2^{271}}$ :  $f(z) = z^{271} + z^{207} + z^{175} + z^{111} + 1$
  - $\mathbb{F}_{2^{271}}$ :  $f(z) = z^{271} + z^{201} + 1$

Since squaring is a linear operation, we have:

$$a(z)^2 = \sum_{i=0}^{m-1} a_i z^{2i} = a_{m-1} z^{2m-2} + \dots + a_2 z^4 + a_1 z^2 + a_0$$

$$a(z) = (a_{m-1} a_{m-2} a_{m-3} \dots a_2 a_1 a_0)$$

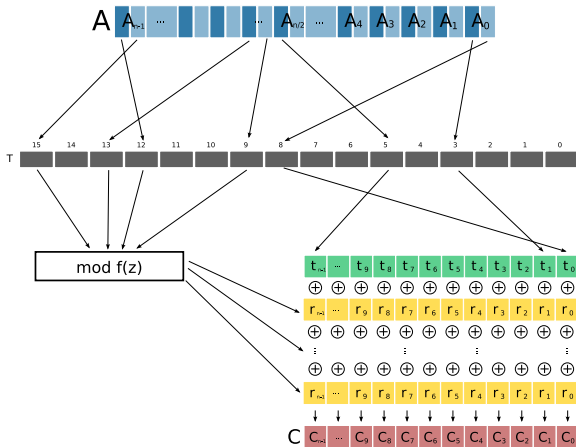
$$a(z)^2 = (a_{m-1} 0 a_{m-2} 0 a_{m-3} 0 \dots 0 a_2 0 a_1 0 a_0)$$

We can accelerate this algorithm with a lookup table.

For each 4-bit  $u$ , compute  $T(u) = (0, u_3, 0, u_2, 0, u_1, 0, u_0)$ :

	0	1	2	3	4	5	6	7
	000000	000001	0000100	0000101	0010000	0010001	0010100	0010101
T	8	9	10	11	12	13	14	15
	0100000	0100001	0100100	0100101	0110000	0110001	0110100	0110101

# Modular squaring in $\mathbb{F}_{2^m}$



---

**Algorithm 1** Modular reduction for  $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$ .

---

**Input:**  $c(z) = c[0..2n - 1]$ .

**Output:**  $c(z) = c(z) \bmod f(z)$ .

```
1: for  $i \leftarrow 41$  to 21 do
2:    $t \leftarrow c[i]$ 
3:    $c[i - 21] \leftarrow c[i - 21] \oplus (t \ll 5)$ 
4:    $c[i - 20] \leftarrow c[i - 20] \oplus (t \ll 4) \oplus (t \gg 3) \oplus t \oplus (t \gg 3)$ 
5:    $c[i - 19] \leftarrow c[i - 19] \oplus (t \gg 4) \oplus (t \gg 5)$ 
6: end for
7:  $t \leftarrow c[20] \gg 3$ 
8:  $c[0] \leftarrow c[0] \oplus (t \ll 7) \oplus (t \ll 6) \oplus (t \ll 3) \oplus t$ 
9:  $c[1] \leftarrow c[1] \oplus (t \gg 1) \oplus (t \gg 2)$ 
10:  $c[20] \leftarrow c[20] \& 0x07$ 
11: return  $c$ 
```

---

**Problems:** Many memory operations and expensive shifts

$R(r_0, r_1, r_2, t)$

1:  $s_0 \leftarrow t \ll 4$

2:  $r_0 \leftarrow (r_0 \oplus t \oplus (t \gg 1)) \gg 4$

3:  $r_1 \leftarrow r_1 \oplus t \oplus (t \ll 3) \oplus s_0 \oplus (t \gg 3)$

4:  $r_2 \leftarrow s_0 \ll 1$

---

---

---

$$R(r_0, r_1, r_2, t)$$

$$1: s_0 \leftarrow t \ll 4$$

$$2: r_0 \leftarrow (r_0 \oplus t \oplus (t \gg 1)) \gg 4$$

$$3: r_1 \leftarrow r_1 \oplus t \oplus (t \ll 3) \oplus s_0 \oplus (t \gg 3)$$

$$4: r_2 \leftarrow s_0 \ll 1$$

---

Optimized version:

---

---

$$R(r_0, r_1, r_2, t)$$

$$1: r_0 \leftarrow r_0 \oplus T_0[t]$$

$$2: r_1 \leftarrow r_1 \oplus T_1[t]$$

$$3: r_2 \leftarrow t \ll 5$$

$T_0, T_1$ : lookup tables of 16 bytes

---

# Proposed optimization for modular reduction

**Our solution:** Small register window and lookup tables.

---

**Algorithm 2** Proposed optimization for modular reduction in  $\mathbb{F}_{2^{163}}$ .

---

**Input:**  $c(z) = c[0..2n - 1]$ ,  $T_0$ ,  $T_1$ .

**Output:**  $c(z) = c(z) \bmod f(z)$ .

**Note:**  $R(r_0, r_1, r_2, t) \equiv r_0 \leftarrow r_0 \oplus T_0[t], r_1 \leftarrow r_1 \oplus T_1[t], r_2 \leftarrow t \ll 5$

```
1:  $r_b \leftarrow 0, r_c \leftarrow 0$ 
2:  $i \leftarrow 21, j \leftarrow 40$ 
3: while  $i > 3$  do
4:    $R(r_b, r_c, r_a, c[j])$ ,  $c[i] \leftarrow c[i] \oplus r_b$ 
5:    $R(r_c, r_a, r_b, c[j - 1])$ ,  $c[i - 1] \leftarrow c[i - 1] \oplus r_c$ 
6:    $R(r_a, r_b, r_c, c[j - 2])$ ,  $c[i - 2] \leftarrow c[i - 2] \oplus r_a$ 
7:    $i \leftarrow i - 3, j \leftarrow j - 3$ 
8: end while
9:  $R(r_b, r_c, r_a, c[22])$ ,  $c[3] \leftarrow c[3] \oplus r_b$ 
10:  $R(r_c, r_a, r_b, c[21])$ ,  $c[2] \leftarrow c[2] \oplus r_c$ 
11:  $c[1] \leftarrow c[1] \oplus r_a$ 
12:  $c[0] \leftarrow c[0] \oplus r_b$ 
13: ...
14: return  $c$ 
```

---

---

**Algorithm 3** Fast reduction for  $f(z) = z^{271} + z^{207} + z^{175} + z^{111} + 1$ .

---

**Input:**  $a(z) = a[0..2n - 1]$ .

**Output:**  $c(z) = a(z) \bmod f(z)$ .

```
1: for  $i \leftarrow 67$  to  $34$  do
2:    $t \leftarrow c[i]$ 
3:    $c[i - 8] \leftarrow c[i - 8] \oplus t$ 
4:    $c[i - 12] \leftarrow c[i - 12] \oplus t$ 
5:    $c[i - 20] \leftarrow c[i - 20] \oplus t$ 
6:    $c[i - 33] \leftarrow c[i - 33] \oplus (t \ggg 7)$ 
7:    $c[i - 34] \leftarrow c[i - 34] \oplus (t \lll 1)$ 
8: end for
9:  $t \leftarrow c[33] \ggg 7$ 
10:  $c[0] \leftarrow c[0] \oplus t$ 
11:  $t \leftarrow c[33] \lll 7$ 
12:  $c[13] \leftarrow c[13] \oplus t$ 
13:  $c[21] \leftarrow c[21] \oplus t$ 
14:  $c[25] \leftarrow c[25] \oplus t$ 
15:  $c[33] \leftarrow c[33] \oplus t$ 
16: return  $c$ 
```

---

---

**Algorithm 4** Fast reduction for  $f(z) = z^{271} + z^{207} + z^{175} + z^{111} + 1$ .

---

**Input:**  $c(z) = c[0..2n - 1]$ .

**Output:**  $c(z) = c(z) \bmod f(z)$ .

```
1: for  $i \leftarrow 67$  to  $34$  do
2:    $t \leftarrow c[i]$ ,  $c[i - 8] \leftarrow c[i - 8] \oplus t$ 
3:    $c[i - 12] \leftarrow c[i - 12] \oplus t$ ,  $c[i - 20] \leftarrow c[i - 20] \oplus t$ 
4: end for
5:  $t \leftarrow a[67]$ ,  $c[33] \leftarrow c[33] \oplus (t \ll 1)$ ,  $c[34] \leftarrow c[34] \oplus (t \gg 7)$ 
6: for  $i \leftarrow 66$  to  $35$  by  $2$  do
7:    $R(r_b, r_a, c[i])$ ,  $c[i - 33] \leftarrow c[i - 33] \oplus r_b$ 
8:    $R(r_a, r_b, c[i - 1])$ ,  $c[i - 34] \leftarrow c[i - 34] \oplus r_a$ 
9: end for
10:  $R(r_b, r_a, c[34])$ ,  $c[1] \leftarrow c[1] \oplus r_b$ ,  $c[0] \leftarrow c[0] \oplus r_a$ 
11: if  $c[33] \wedge 0x80$  then
12:    $c[0] \leftarrow c[0] \oplus 0x01$ ,  $c[13] \leftarrow c[13] \oplus 0x80$ 
13:    $c[21] \leftarrow c[21] \oplus 0x80$ ,  $c[25] \leftarrow c[25] \oplus 0x80$ 
14:    $c[33] \leftarrow c[33] \wedge 0x7F$ 
15: end if
16: return  $c$ 
```

---

$$c(z) = \sum_{i=0}^{m-1} c_i z^i; \quad \sqrt{c(z)} = ?$$

Since squaring is a linear operation:

$$\begin{aligned} \sqrt{c(z)} &= \sum_{i \text{ even}} c_i z^{\frac{i}{2}} + \sqrt{z} \sum_{i \text{ odd}} c_i z^{\frac{i-1}{2}} \\ &= C_{\text{even}} + \sqrt{z} \cdot C_{\text{odd}} \end{aligned}$$

---

**Algorithm 5** Square-root in  $\mathbb{F}_{2^{271}}$ . [Fong et al. 2004, Avanzi 2007, Scott]

---

**Input:**  $a(z) = a[0..n - 1]$ ,  $evens[16]$ ,  $odds[16]$ .

**Output:**  $c(z) = \sqrt{a(z)}$ .

**Note:**  $\sqrt{z} = z^{136} + z^{104} + z^{88} + z^{56}$ .

```
1:  $c \leftarrow 0$ 
2: for  $i \leftarrow 0$  to 34 do
3:    $x \leftarrow evens[c[i]]$ 
4:    $y \leftarrow odds[c[i]]$ 
5:    $x \leftarrow x \vee evens[c[i + 1]]$ 
6:    $y \leftarrow y \vee odds[c[i + 1]]$ 
7:    $c[i/2] \leftarrow c[i/2] \oplus x$ 
8:    $c[i/2 + 17] \leftarrow y$ 
9:    $c[i/2 + 13] \leftarrow c[i/2 + 13] \oplus y$ 
10:   $c[i/2 + 11] \leftarrow c[i/2 + 11] \oplus y$ 
11:   $c[i/2 + 7] \leftarrow c[i/2 + 7] \oplus y$ 
12: end for
13: return  $c$ 
```

---

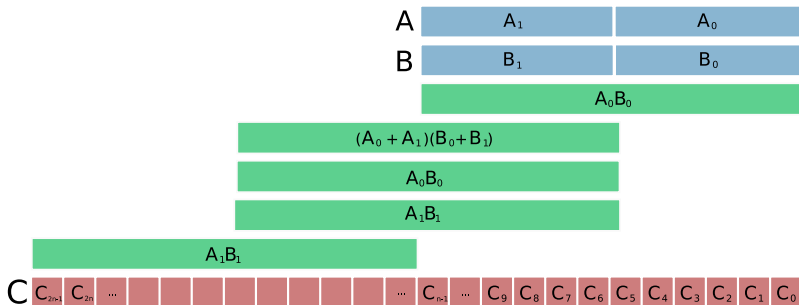
**Problem:** Redundant memory operations!

**Our solution:** Accumulate writes in registers!

- Step 1: Multiplication
  - Karatsuba multiplication;
  - Comb multiplication (López-Dahab, 1999);
- Step 2: Modular reduction

# Karatsuba multiplication in $\mathbb{F}_{2^m}$

$$\begin{aligned}
 c(z) &= a(z) \cdot b(z) \\
 &= a_1 b_1 z^m + [(a_1 + a_0)(b_1 + b_0) + a_1 b_1 + a_0 b_0] z^{m/2} + a_0 b_0
 \end{aligned}$$

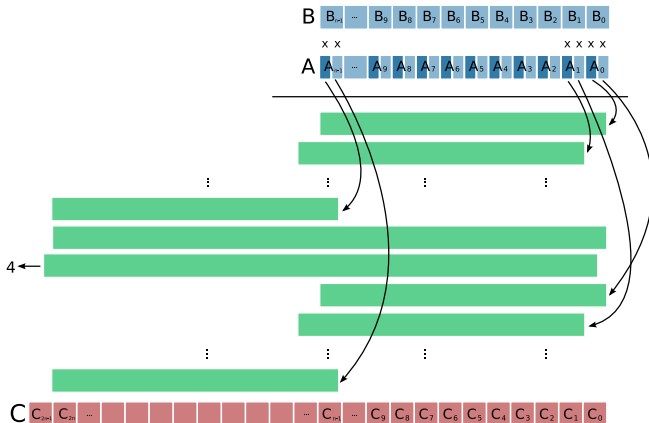


# Comb multiplication in $\mathbb{F}_{2^m}$

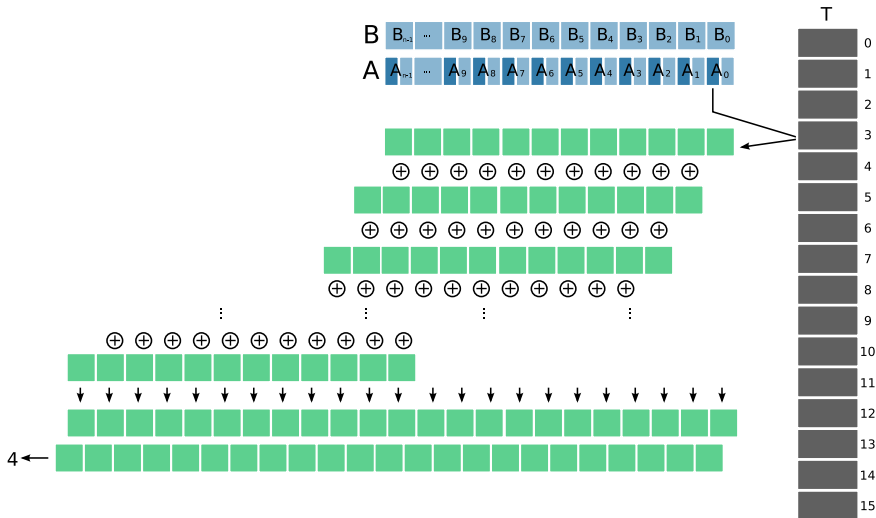
This method pre-computes  $u(z) \cdot b(z)$  for all polynomials  $u(z)$  of degree less than 4.

$$\square \times \begin{matrix} B_{r-1} & \dots & B_9 & B_8 & B_7 & B_6 & B_5 & B_4 & B_3 & B_2 & B_1 & B_0 \end{matrix}$$

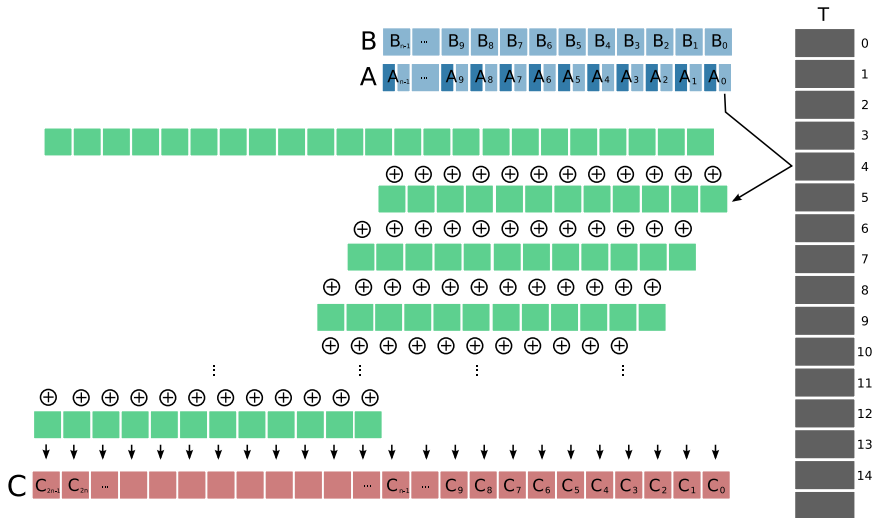
If  $a(z)$  is divided into 4-bit polynomials, compute  $a(z) \cdot b(z)$  by:



# Comb multiplication in $\mathbb{F}_{2^m}$



# Comb multiplication in $\mathbb{F}_{2^m}$

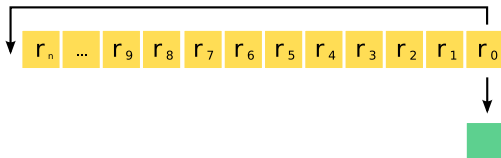


**Problem:** Many memory operations and not enough registers!

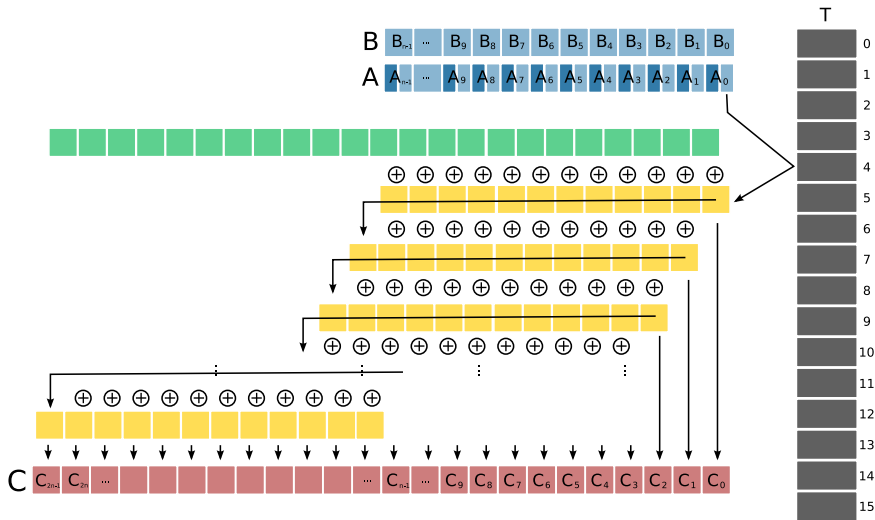
**Our solution:** Use a rotating register window of length  $n + 1$

# Comb multiplication: registers

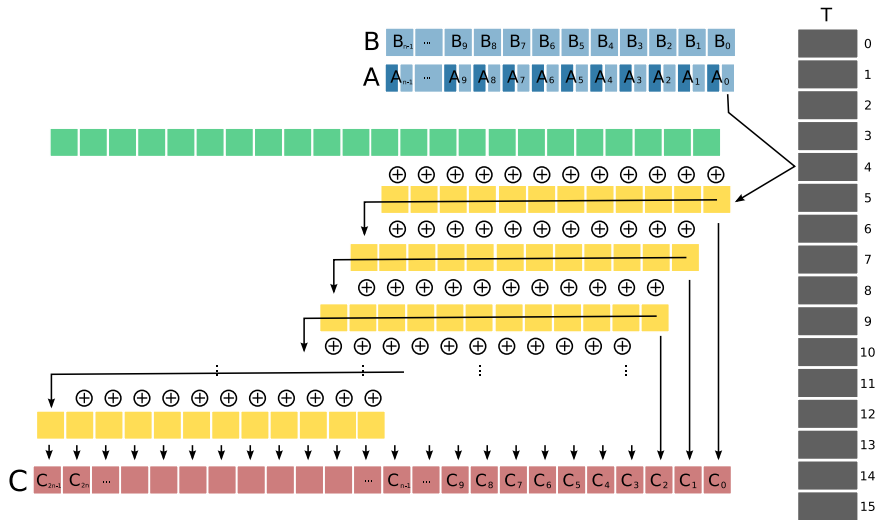
**Our solution:** Use a rotating register window of length  $n + 1$



# Proposed optimization for Comb multiplication



# Proposed optimization for Comb multiplication



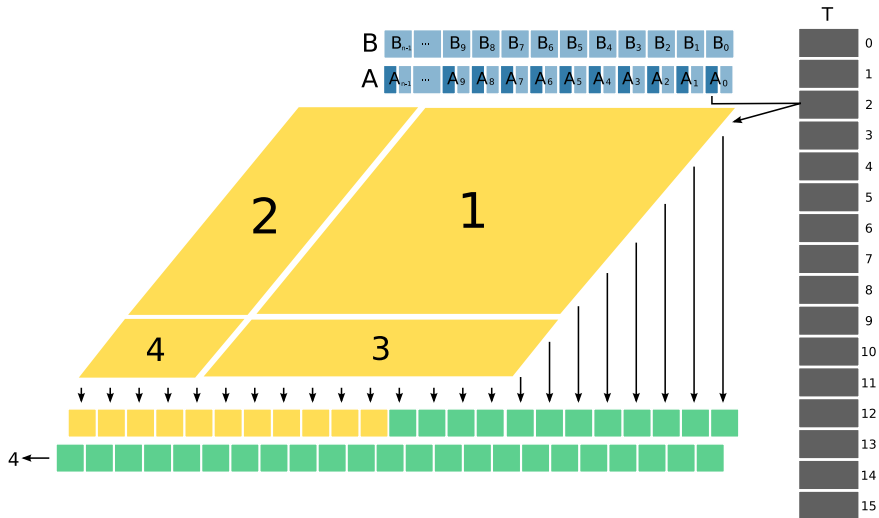
**Problem:** Available registers might be insufficient ( $\mathbb{F}_{2^{233}}, \mathbb{F}_{2^{271}}$ ).

# Block implementation of Comb multiplication

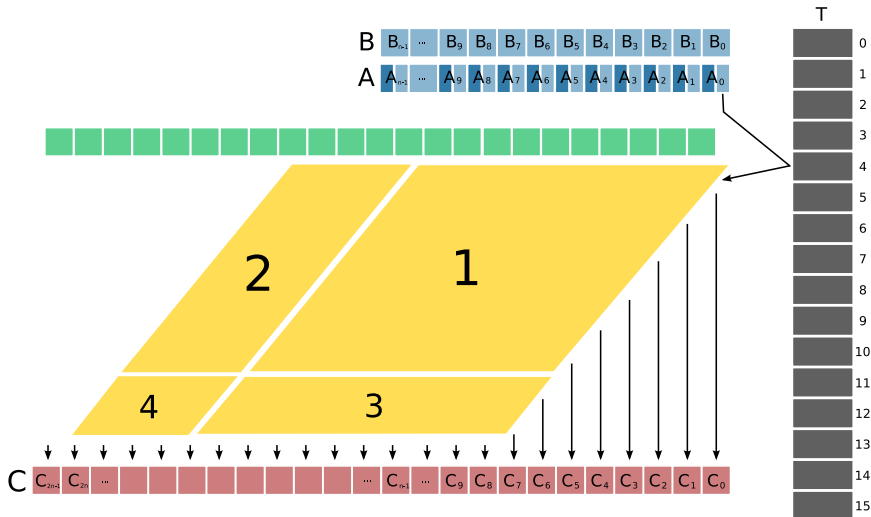
**Our solution:** Break down the summations in blocks.

# Block implementation of Comb multiplication

**Our solution:** Break down the summations in blocks.



# Block implementation of Comb multiplication



## Timings for ATmega128L

# ATmega128L: Implementation

- GCC 4.1.2 for ATmega128L;
- Software library implemented from scratch;
- AVR Simulator 4.14.

Programming languages:

- C;
- Assembly.

Curve parameters:

- Koblitz curves NIST-K163 and NIST-K233;
- Binary curves NIST-B163 and NIST-B233.

ECC protocols:

- ECDSA

	C language	<i>Assembly</i>
<b>Algorithm</b>	Cycles	Cycles
Modular Squaring	1154	570
LD Mult. with registers	<b>9738</b>	<b>4508</b>
Karatsuba+LD with registers	12246	6968
Modular reduction	606	430
Inversion	243790	81365

**Table:** Timings for arithmetic algorithms in  $\mathbb{F}_{2^{163}}$ .

$$n = \lceil 163/8 \rceil = 21$$

	C language	Assembly
<b>Algorithm</b>	Cycles	Cycles
Modular Squaring	1340	956
LD Mult. with registers (blocks)	<b>18028</b>	<b>8314</b>
Karatsuba+LD with registers	25850	9261
Modular reduction	911	620
Inversion	473618	142986

**Table:** Timings for arithmetic operations in  $\mathbb{F}_{2^{233}}$ .

$$n = \lceil 233/8 \rceil = 30$$

	C language	Assembly
Algorithm	Cycles	Cycles
Squaring	2639	1429
Square-root	2613	1361
LD Mult. with registers (blocks)	26019	10812
Karatsuba+LD with registers	29652	11393
Modular reduction	1713	1059

Table: Timings for arithmetic operations in  $\mathbb{F}_{2^{271}}$ .

$$n = \lceil 271/8 \rceil = 34$$

## Koblitz curves

	K-163	K-233
<b>Algorithms</b>	Time (s)	Time (s)
Key Generation	0.29	0.66
Signing	0.36	0.78
Verification	0.63	1.39

## Random curves

	B-163	B-233
<b>Algorithms</b>	Time (s)	Time (s)
Key Generation	0.37	0.94
Signing	0.45	1.04
Verification	1.04	2.55

# Bilinear pairings for binary elliptic curves

- A supersingular binary curve:  $y^2 + y = x^3 + x + \mathbf{b}$
- The **order** of this curve is  $N = 2^m + 1 \pm 2^{\frac{m+1}{2}}$
- The **embedding degree** is  $k = 4$  (the least integer such that  $N$  divides  $2^{km} - 1$ ).
- Choosing  $T = 2^m - N$  and a prime  $r$  dividing  $N$ , [Barreto et al. 2004] defined the reduced  $\eta_T$  pairing:

$$\begin{aligned}\eta_T &: E(\mathbb{F}_{2^m})[r] \times E(\mathbb{F}_{2^m})[r] \rightarrow \mathbb{F}_{2^{4m}}^* \\ \eta_T(P, Q) &= f_{T', P'}(\psi(Q))^{\frac{2^{4m}-1}{N}},\end{aligned}$$

where  $T' = \pm T$  and  $P' = \pm P$ . The function  $f$  is a **Miller function** and  $\psi$  is the **distortion map**  
 $\psi(x, y) = (x^2 + s, y + sx + t)$ .

Let  $q = 2^{271}$ . The extension field  $\mathbb{F}_{q^4}$  is represented using tower extensions  $\mathbb{F}_{q^2} = \mathbb{F}_q[s]/(s^2 + s + 1)$  and  $\mathbb{F}_{q^4} = \mathbb{F}_{q^2}[t]/(t^2 + t + s)$ , and a basis for  $\mathbb{F}_{q^4}$  over  $\mathbb{F}_q$  is  $\{1, s, t, st\}$ .

Multiplication in  $\mathbb{F}_{q^4}$ :

- Karatsuba technique requires 9 base field multiplications and 20 additions;
- Product of sparse elements costs less.

---

**Algorithm 6**  $\eta_T$  pairing [Barreto et al. 2004], [Beuchat et al. 2008].

---

**Input:**  $P = (x_P, y_P), Q = (x_Q, y_Q) \in E(\mathbb{F}_{2^m})[r]$ .

**Output:**  $\eta_T(P, Q) \in \mathbb{F}_{2^{4m}}^*$ .

```
1:  $y_P \leftarrow y_P + 1 - \delta$ 
2:  $u \leftarrow x_P + \alpha, v \leftarrow x_Q + \alpha$ 
3:  $g_0 \leftarrow u \cdot v + y_P + y_Q + \beta$  (1 M, 2 A)
4:  $g_1 \leftarrow u + x_Q, g_2 \leftarrow v + x_P^2$  (1 S, 2 A)
5:  $G \leftarrow g_0 + g_1s + t$ 
6:  $L \leftarrow (g_0 + g_2) + (g_1 + 1)s + t$  (1 A)
7:  $F \leftarrow L \cdot G$  (2 M, 1 S, 5 A)
8: for  $i \leftarrow 1$  to  $\frac{m-1}{2}$  do
9:    $x_P \leftarrow \sqrt{x_P}, y_P \leftarrow \sqrt{y_P}, x_Q \leftarrow x_Q^2, y_Q \leftarrow y_Q^2$  (2 R, 2 S)
10:   $u \leftarrow x_P + \alpha, v \leftarrow x_Q + \alpha$ 
11:   $g_0 \leftarrow u \cdot v + y_P + y_Q + \beta$  (1 M, 2 A)
12:   $g_1 \leftarrow u + x_Q$  (1 A)
13:   $G \leftarrow g_0 + g_1s + t$ 
14:   $F \leftarrow F \cdot G$  (6 M, 14 A)
15: end for
16: return  $F^{(2^{2m}-1)(2^{m+1} \pm 2^{\frac{m+1}{2}})}$  (26 M, (2m + 9) S, (2m + 53) A, 1)
```

---

Field representation:

- $\mathbb{F}_{2^{271}} = \mathbb{F}_2[x]/x^{271} + x^{207} + x^{175} + x^{111} + 1.$

Curve parameters:

- Pairing-friendly supersingular curve  $y^2 + y = x^3 + x$  from MIRACL. This curve has a small cofactor  $h = 487805$ .

	C language	Assembly
<b>Work</b>	Time (s)	Time (s)
NanoECC-2008	10.96	-
TinyPBC-2008	5.45	-
NanoECC-SKSC-2009*	10.91	2.66
<b>Our NanoPBC-2009</b>	<b>4.44</b>	<b>2.06</b>

Table: Timings for  $\eta_T$  pairing in  $\mathbb{F}_{2^{271}}$ .

SKSC: Piotr Szczechowiak, Anton Kargl, Michael Scott, and Martin Collier, 2009.

	ROM memory	RAM memory
NanoECC (C)	53 KB	–
TinyPBC (C)	48 KB	0.4 KB
NanoECC-SKSC (ASM)	61 KB	–
<b>NanoPBC (C-only)</b>	19 KB	0.4 KB
<b>NanoPBC (C+ASM)</b>	24 KB	0.4 KB

Table: Cost in *bytes* of memory for implementations of the  $\eta_T$  pairing.

## Intel PXA270 processor

# The micro-controller Intel PXA270 processor

- Intel Xscale microarchitecture
- 32-bit ARMv5TE clocked at 520 MHz
- 16 32-bit register purpose registers
- Used on wireless sensors (commonly use in smartphones)
- Intel Wireless Technology:
  - SIMD instructions for multimedia applications
  - MMX technology and Intel's SSE instruction set
  - 16 64-bit registers
  - 4 32-bit control registers
- Memory: 32 KB data cache, 32 KB instruction cache
- Shifter in most data-processing instructions  $c = a \oplus (b \ll k)$

- Binary field arithmetic over  $\mathbb{F}_{2^m}$ ,  $m = 163, 271, 283$
- Elliptic curve implementation B-163, B-283
- Pairing computation: the  $\eta_T$  pairing for  $E/\mathbb{F}_{2^{271}}$
- Assembly code (GPR-32, WMMX-64)

	GPR-32	WMMX-64
<b>Algorithm</b>	Cycles	Cycles
Squaring	187	–
Square-root	185	–
Multiplication (LD)	2025	1411

**Table:** Timings for arithmetic operations in  $\mathbb{F}_{2^{271}}$ .

	GPR-32	WMMX-64
<b>Algorithm</b>	Cycles	Cycles
Squaring	231	–
Square-root	432	–
Multiplication (LD)	2119	1476
Inversion	22198	12831
Q. Equation	1078	–

**Table:** Timings for arithmetic operations in  $\mathbb{F}_{2^{283}}$ .

	GPR-32	WMMX-64
<b>Algorithm</b>	cycles $\times 10^6$	cycles $\times 10^6$
$kP$ (Comb)	0.37	0.28
$kP$ (Halving)	0.78	0.61
$kP + sQ$	1.02	0.78

Table: Timings for point multiplication NIST B-163 curve.

	GPR-32	WMMX-64
<b>Algorithm</b>	cycles $\times 10^6$	cycles $\times 10^6$
$kP$ (Comb)	1.37	0.99
$kP$ (Halving)	2.66	1.96
$kP + sQ$	3.80	2.45

Table: Timings for point multiplication NIST B-283 curve.

	GPR-32	WMMX-64
<b>Work</b>	cycles $\times 10^6$	cycles $\times 10^6$
NanoECC-SKSC-2009*	6.0	—
<b>Our work</b>	2.5	1.9

Table: Timings for  $\eta_T$  pairing in  $\mathbb{F}_{2^{271}}$ .

\*ARM PXA271

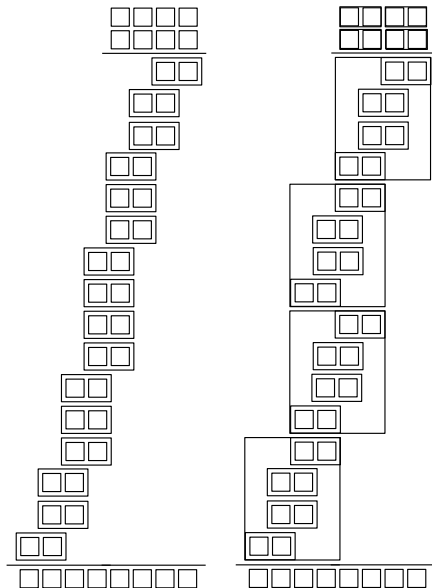
MPS430 processor

# The micro-controller MSP430

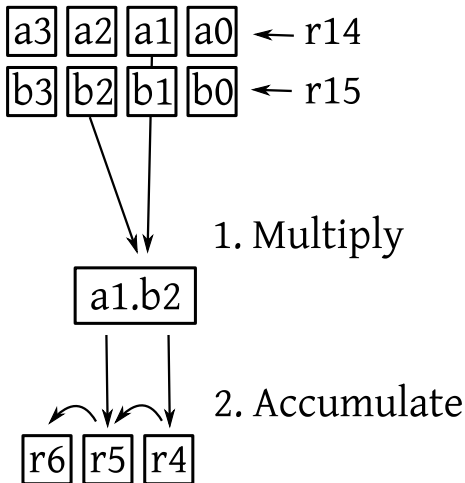
- Used on wireless sensors
  - Tmote Sky from Moteiv
  - TelosB from Crossbow
- 12 general purpose registers
- Memory: 10 KB RAM, 48 KB ROM
- Instructions
  - 27 instructions
  - register-register, register-memory, memory-memory
  - One bit only shifts
- Multiplier is a memory mapped peripheral

- Multi-precision Multiplication: Comba method, Hybrid method
- Multiplier (4 options)
  - (Un)signed multiply
  - (Un)signed multiply and accumulate
- Reduction: Montgomery, Barrett
- Main optimization: the use of MAC (Multiply and Accumulate) operation

# Multiplication methods: Comba and Hybrid



# Comba multiplication using MAC



- GCC 3.23 for MSP430;
- Software library implemented from scratch;
- MSPSIM Simulator SVN

Programming languages:

- C;
- Assembly.

Curve parameters:

- Random curves SECG-160r1 and NIST-P-256,

ECC protocols:

- ECDSA

Algorithm	Cycles
<i>256 bits:</i>	
Mult. (Hybrid-K-128)	4025
Mult. (Comba MAC-K-128)	3597
Mult. (Comba MAC-256)	3689
Squaring (Comba MAC-K-128)	2960
<i>160 bits:</i>	
Mult. (Comba MAC-160)	1586
Squaring (Comba MAC-160)	1371

Table: Timings for multiplication:160 and 256 bits

Algorithm	Cycles
<i>256 bits:</i>	
Montgomery	4761
Montgomery MAC	3989
Barrett	4773
NIST $p$ -256	709
<i>160 bits:</i>	
Montgomery MAC	1785
SECG $p$ -160	342

Table: Timings for modular reduction

Algorithm	Cycles
<i>256 bits:</i>	
Mult. (Hybrid-K-128)	8855
Mult. (Comba MAC-K-128)	7604
Squaring (Comba MAC-K-128)	6952
<i>160 bits:</i>	
Mult. (Hybrid-[SKSC-2009])	4734
Mult. (Comba MAC-160)	<b>3389</b>
Squaring (Comba MAC-160)	3172

**Table:** Timings for field multiplication: using Montgomery reduction

	SECG-160r1	NIST P-256
<b>Algorithms</b>	Time (s)	Time (s)
Key Generation	0.23	0.71
Signing	0.27	0.75
Verification	0.69	2.02

Table: Timings for ECDSA

- Parametric family of curves with embedding degree 12
- Modulus:  $p(x) = 36x^2 + 36x^3 + 24x^2 + 6x + 1$
- Order:  $E(\mathbb{F}_p) : n(x) = 36x^2 + 36x^3 + 18x^2 + 6x + 1$
- Frobenius trace:  $t(x) = 6x^2 + 1$
- Existence of a sextic twist
  - Normal pairing:  $E(\mathbb{F}_p) \times E(\mathbb{F}_{p^{12}}) \rightarrow \mathbb{F}_{p^{12}}$
  - Pairing with twist:  $E(\mathbb{F}_p) \times E'(\mathbb{F}_{p^2}) \rightarrow \mathbb{F}_{p^{12}}$

- Tate ( $r = n(x)$ ; 256 bits) [Tate, 1957]
- Ate ( $r = t(x) - 1$ ;  $\sim 128$  bits) [Hess et al, 2006]
- Optimal Ate ( $r = 6x + 2$ ;  $\sim 64$  bits) [Vercauteren 2008]
- R-ate ( $r = 6x + 2$ ;  $\sim 64$  bits) [Eunjeong et al, 2008]
- Xate ( $r = x$ ;  $\sim 64$  bits) [Nogami et al, 2008]

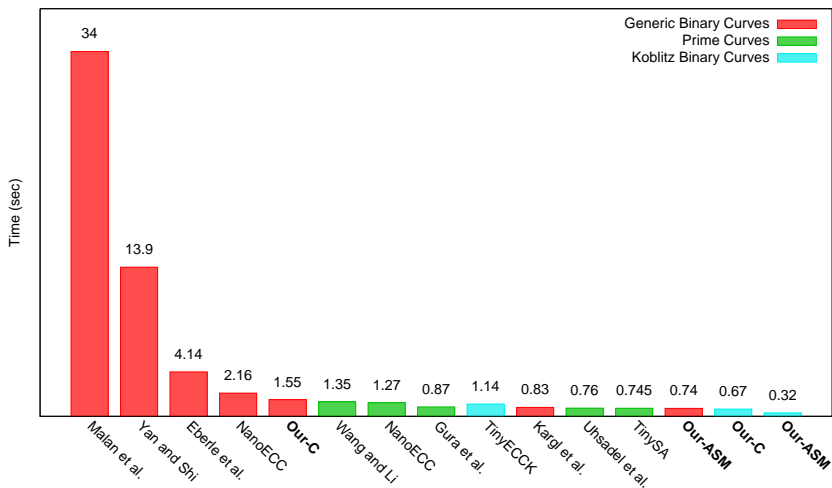
Algorithm	Time (s)
Optimal Ate	16.8
R-ate	16.8
Xate	16.6

Table: Timings for pairing computation on BN curve ( $k = 12$ , 256 bits)

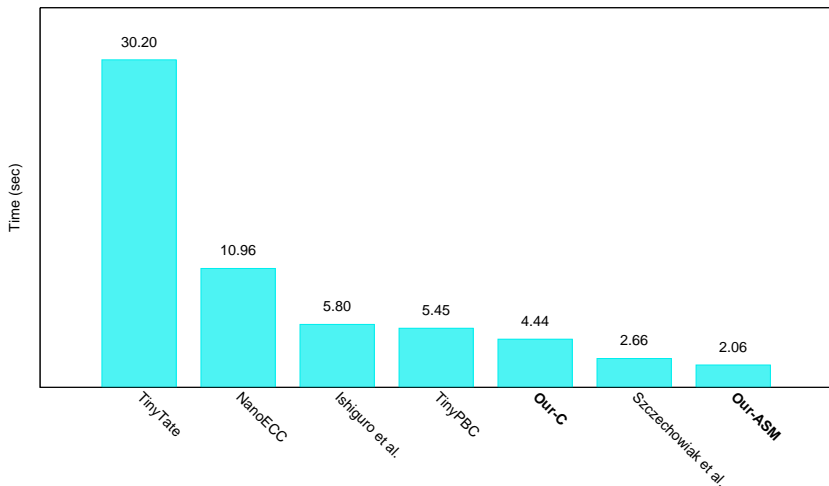
Algorithm	Time (s)
Tate, -O0 (SKSC 2009)	4.72
Our Tate, -O0 (MAC)	3.92
Our Tate, -O2 (MAC)	3.29

Table: Timings for pairing computation on MNT curve ( $k = 4$ , 160 bits)

# ATmega128L: Performance of point multiplication at 80-bit security level



# ATmega128L: Performance of paring computation at 80-bit security level



# Conclusions

New *state of the art* in software implementation of ECC and pairings for WSNs:

New *state of the art* in software implementation of ECC and pairings for WSNs:

- Efficient implementation of field arithmetic:  $\mathbb{F}_p$ ,  $\mathbb{F}_{2^m}$

New *state of the art* in software implementation of ECC and pairings for WSNs:

- Efficient implementation of field arithmetic:  $\mathbb{F}_p$ ,  $\mathbb{F}_{2^m}$
- Efficient implementation of elliptic curve cryptography: ECDSA (**new records**)

New *state of the art* in software implementation of ECC and pairings for WSNs:

- Efficient implementation of field arithmetic:  $\mathbb{F}_p$ ,  $\mathbb{F}_{2^m}$
- Efficient implementation of elliptic curve cryptography: ECDSA (**new records**)
  - Signing and verification under 1 second at 80-bit security level for ATmega128L (Koblitz curves) and MSP430 (prime random curves)

New *state of the art* in software implementation of ECC and pairings for WSNs:

- Efficient implementation of field arithmetic:  $\mathbb{F}_p$ ,  $\mathbb{F}_{2^m}$
- Efficient implementation of elliptic curve cryptography: ECDSA (**new records**)
  - Signing and verification under 1 second at 80-bit security level for ATmega128L (Koblitz curves) and MSP430 (prime random curves)
  - Signing and verification under 2 seconds at 128-bit security level for MSP430 (prime random curves).

New *state of the art* in software implementation of ECC and pairings for WSNs:

- Efficient implementation of field arithmetic:  $\mathbb{F}_p$ ,  $\mathbb{F}_{2^m}$
- Efficient implementation of elliptic curve cryptography: ECDSA (**new records**)
  - Signing and verification under 1 second at 80-bit security level for ATmega128L (Koblitz curves) and MSP430 (prime random curves)
  - Signing and verification under 2 seconds at 128-bit security level for MSP430 (prime random curves).

- Efficient implementation of pairing computation for WSNs:  
(new records)

- Efficient implementation of pairing computation for WSNs:  
(new records)
  - Calculating the  $\eta_T$  pairing in 2 seconds at 80-bit security level for ATmega128L.

- Efficient implementation of pairing computation for WSNs:  
(new records)
  - Calculating the  $\eta_T$  pairing in 2 seconds at 80-bit security level for ATmega128L.
  - Calculating the  $\eta_T$  pairing in 1.9 millions of cycles (142 ms at 13 MHz) at 80-bit security level for Intel PXA270.

- Efficient implementation of pairing computation for WSNs:  
(new records)
  - Calculating the  $\eta_T$  pairing in 2 seconds at 80-bit security level for ATmega128L.
  - Calculating the  $\eta_T$  pairing in 1.9 millions of cycles (142 ms at 13 MHz) at 80-bit security level for Intel PXA270.
  - Performance of BN curves at 128-bit security level for MSP430.

- Efficient implementation of pairing computation for WSNs: (new records)
  - Calculating the  $\eta_T$  pairing in 2 seconds at 80-bit security level for ATmega128L.
  - Calculating the  $\eta_T$  pairing in 1.9 millions of cycles (142 ms at 13 MHz) at 80-bit security level for Intel PXA270.
  - Performance of BN curves at 128-bit security level for MSP430.
- More research work is needed to develop security solutions based on Pairing-based cryptography for WSNs.

Thanks for your attention. Any questions?