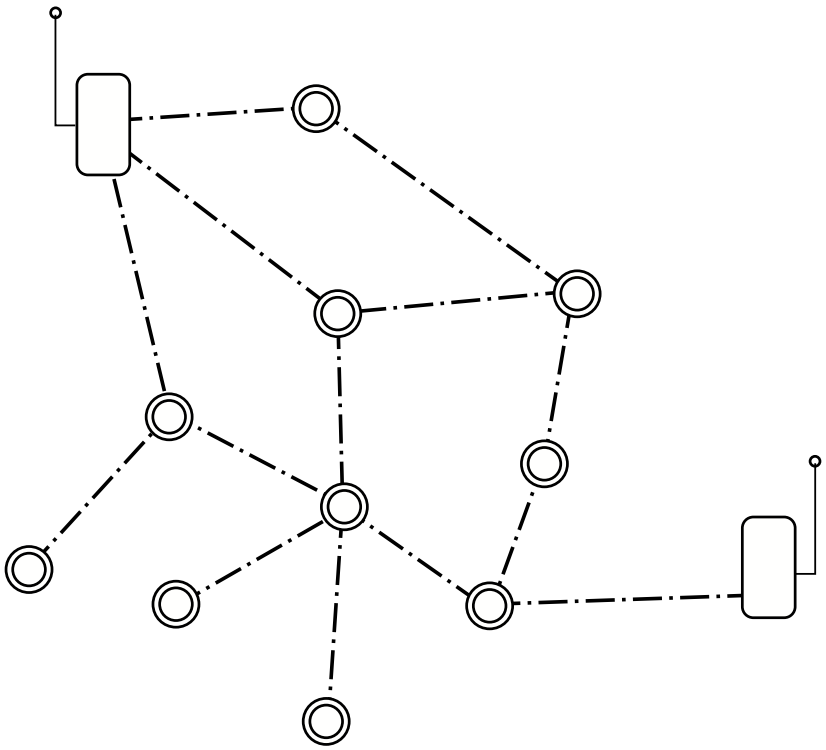
Implementação eficiente de criptografia de curvas elípticas em sensores sem fio

Diego Aranha, Danilo Câmara, Julio López,
Leonardo Oliveira, Ricardo Dahab

Instituto de Computação - UNICAMP
Financiado por FAPESP, processo 2007/06950-0.

Redes de sensores sem fio

Uma rede de sensores sem fio é uma rede *ad hoc* de dispositivos configurados para realizar monitoramento cooperativo.



O problema

Desafio

Proteger uma rede de sensores sem fio, onde os nós são escassos em recursos e de natureza descartável.

Contribuições

- Implementação eficiente de aritmética no corpo $\mathbb{F}_{2^{163}}$;
- Implementação eficiente de criptografia de curvas elípticas.

A plataforma

Características do MICAz Mote:

- Processador ATMega128, 7.3828 MHz;
- 4KB memória RAM, 128KB memória ROM;
- *Pipeline* de 2 estágios;
- Alto custo de instruções de acesso à memória.

Trabalhos relacionados

Tabela: Tempos publicados para a multiplicação de ponto em um MICAz Mote para o nível de segurança de 160 *bits*.

Corpo	Trabalho	Tempo de execução (s)
Binário	[Malan et al. 2004]	34
	[Yan and Shi 2006]	13.9
	[Eberle et al. 2005]	4.14
	[Szczechowiak et al. 2008]	2.16
	[Seo et al. 2008]	1.14
Primo	[Wang and Li 2006]	1.35
	[Szczechowiak et al. 2008]	1.27
	[Gura et al. 2004]	0.81
	[Uhsadel et al. 2007]	0.76
	[GrobSchadl 2006]	0.745

Frases para lembrar...

"*In this paper we consider only prime integer fields since binary polynomial field arithmetic, specifically multiplication, is* **insuficiently supported** *by current microprocessors and would thus lead to* **lower performance.**"

"*This is not the case for arithmetic operations over fields GF(2^m). Though these operations could be implemented in hardware rather efficiently, executing them on standard processors is* **prohibitively slow.**"

Curvas elípticas

Uma *curva elíptica binária* é o conjunto de soluções $(x, y) \in \mathbb{F}_{2^m} \times \mathbb{F}_{2^m}$ que satisfazem a equação

$$y^2 + xy = x^3 + ax^2 + b,$$

onde $a, b \in \mathbb{F}_{2^m}$ com $b \neq 0$, e um *ponto no infinito* ∞.

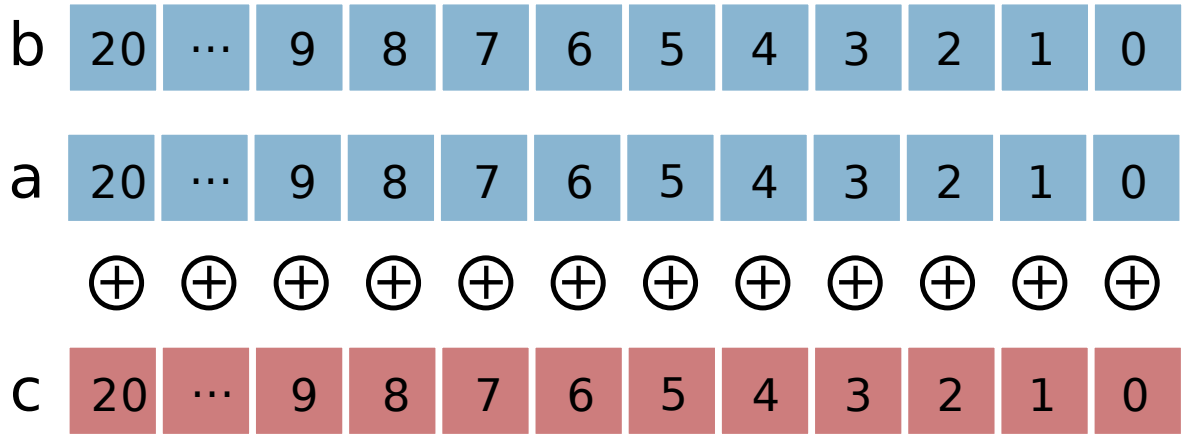
O conjunto de pontos sob a operação $+$ (secante e tangente) forma um grupo aditivo. A *multiplicação de ponto* é definida pela relação de recorrência:

$$kP = \begin{cases} \infty, & \text{se } k = 0; \\ (-k)(-P) & \text{se } k \leq -1; \\ (k-1)P + P & \text{se } k \geq 1. \end{cases}$$

Corpo finito $\mathbb{F}_{2^m}$

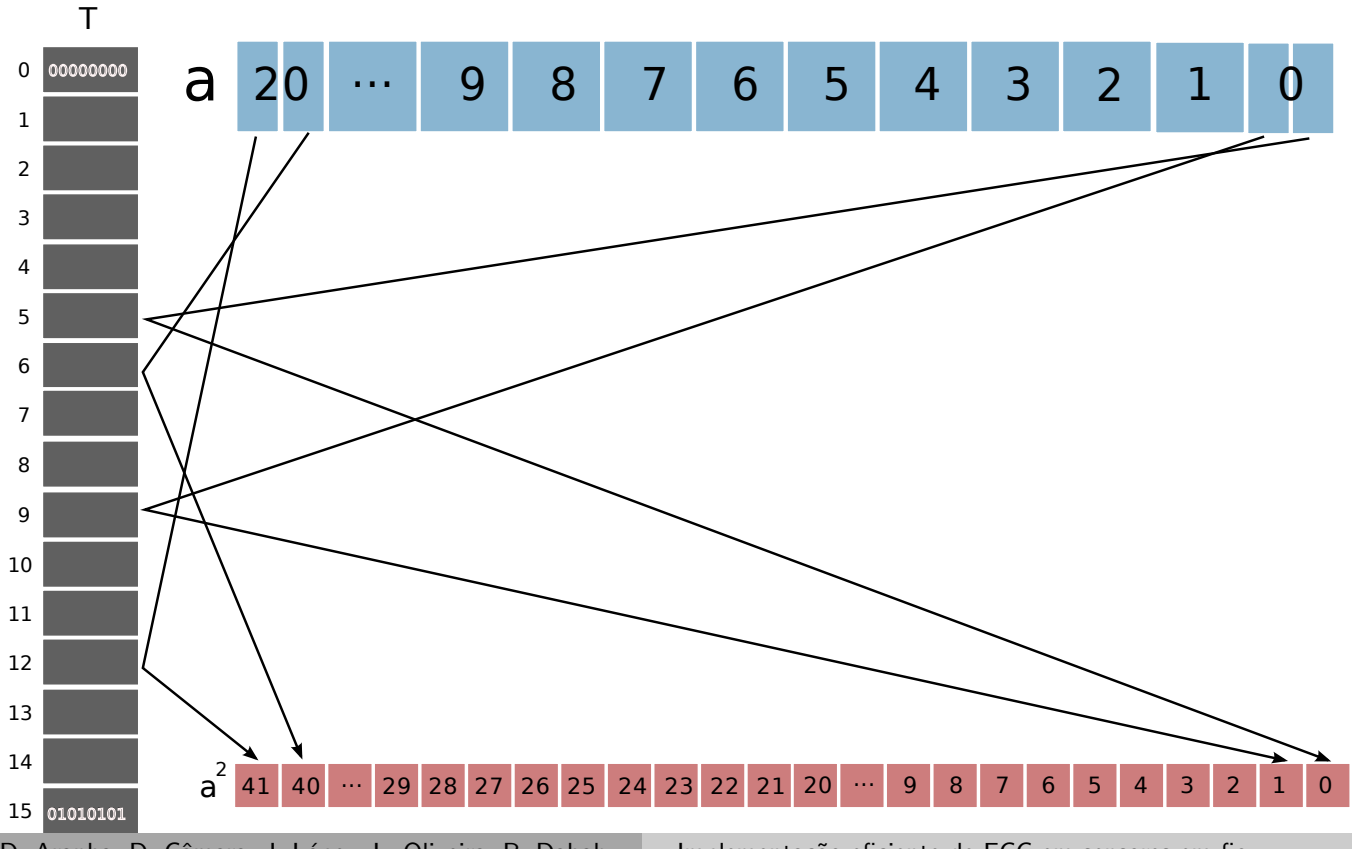
- Base polinomial: $a(z) \in \mathbb{F}_{2^m} = \sum_{i=0}^{m-1} a_i z^i$.
- Representação em *software*: vetor de $\lceil m/8 \rceil$ *bytes*.

Adição em $\mathbb{F}_{2^m}$



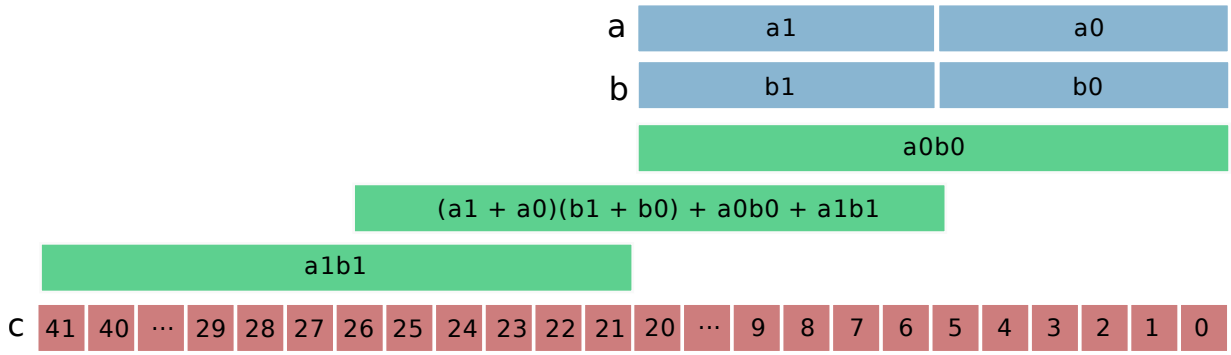
Quadrado em $\mathbb{F}_{2^m}$

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

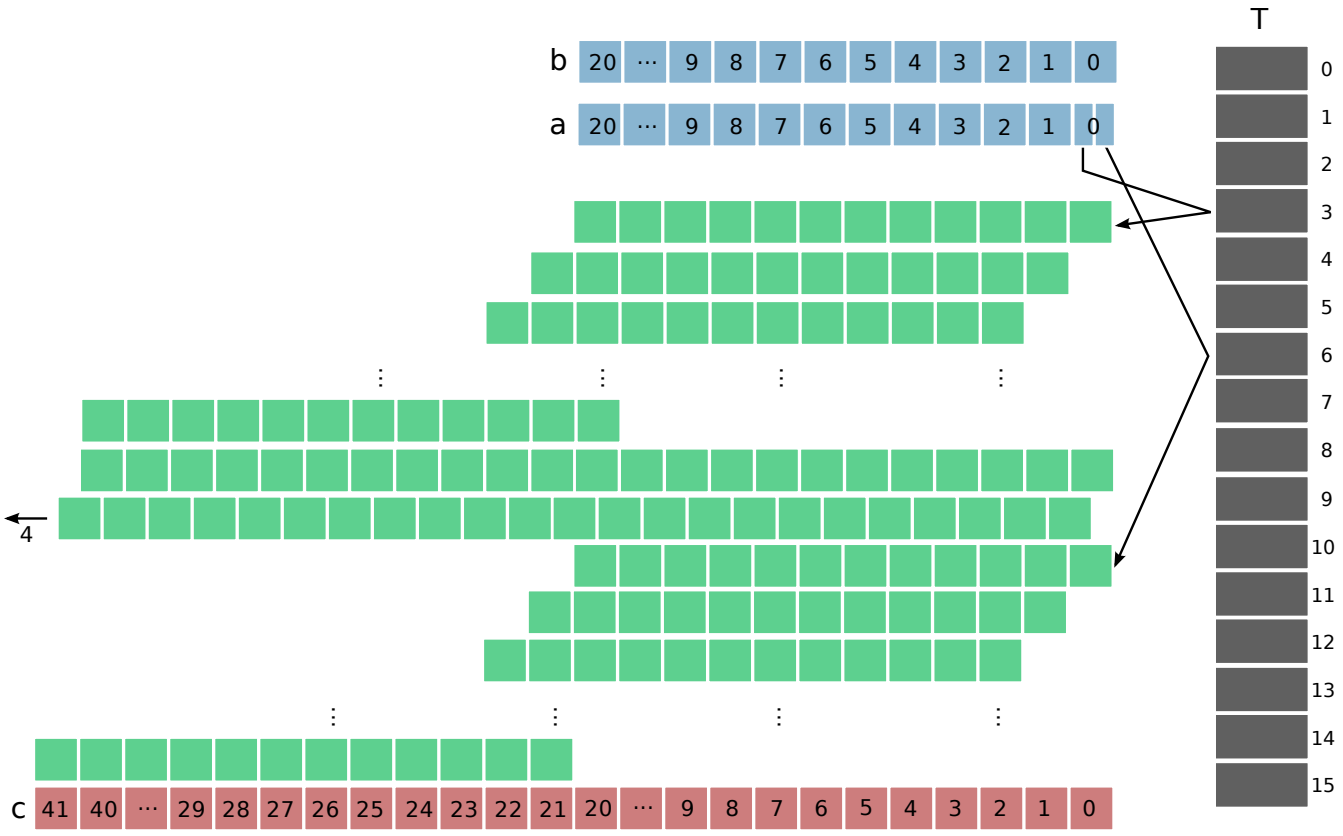


Multiplicação Karatsuba em $\mathbb{F}_{2^m}$

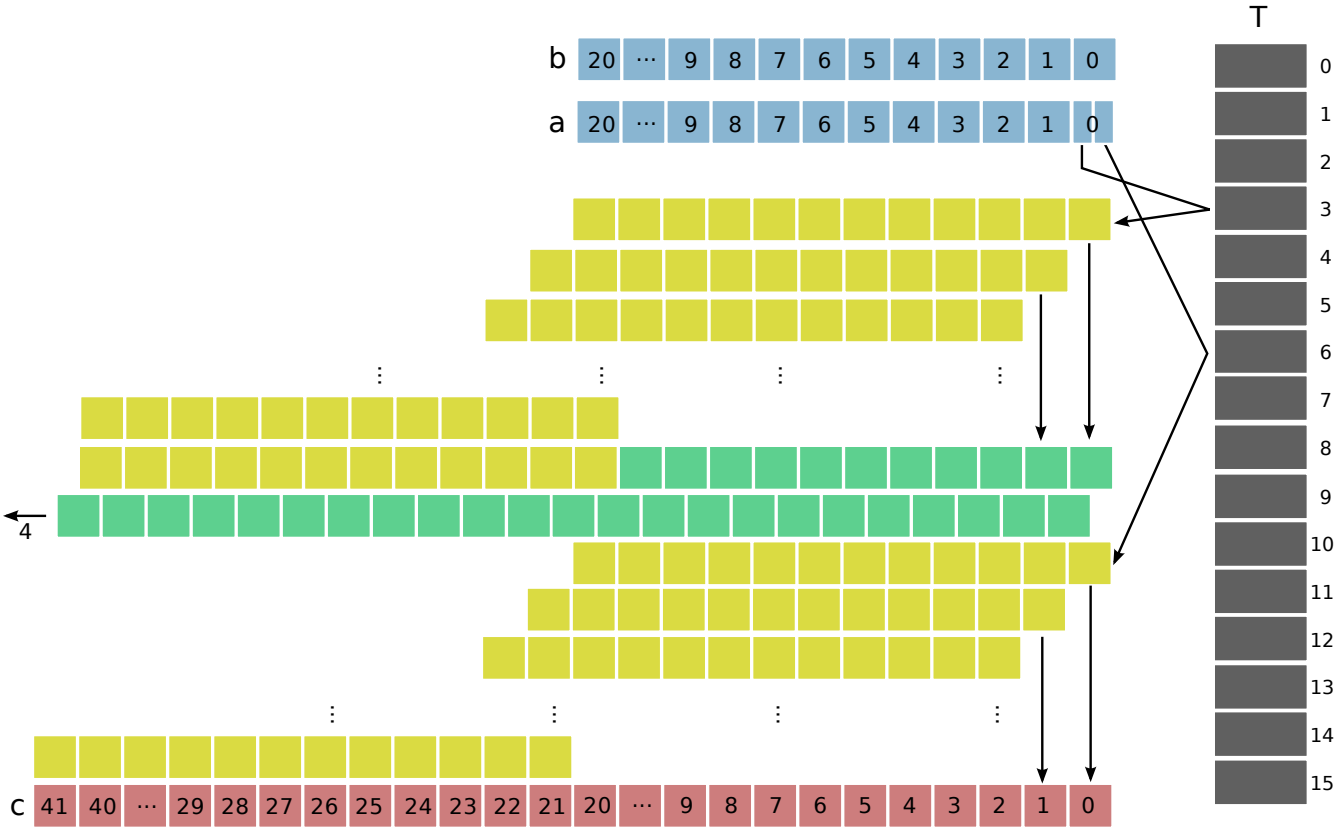
$$a(z)b(z) = a1b1z^m + [(a1+a0)(b1+b0)+a1b1+a0b0]z^{m/2} + a0b0$$



Multiplicação López-Dahab em $\mathbb{F}_{2^m}$



Otimização proposta para Multiplicação López-Dahab



Análise da multiplicação em $\mathbb{F}_{2^m}$

Tabela: Custo em instruções dos algoritmos de multiplicação em $\mathbb{F}_{2^m}$.

	Instruções em função do número de palavras n		
Alg.	Leituras	Escritas	XOR
LD	$4n^2 + 9n - 1$	$2^t n + 2n^2 + 6n - 2$	$2n^2 + 13n$
LD reg.	$2n^2 + 4n - 1$	$2^t n + 5n - 1$	$2n^2 + 11n$
Karat+M	$11n + 3\mathrm{M}(n/2)$	$7n + 3\mathrm{M}(n/2)$	$4n + 3\mathrm{M}(n/2)$

	Instruções em função de $n = 21$		
Algoritmo	Leituras	Escritas	XOR
López-Dahab	1952	1342	1155
LD com registradores	965	440	1113
Karatsuba+LD	1977	1593	1239
Karatsuba+LD com reg.	1086	837	1173

Redução modular

Algoritmo 1 Redução rápida para $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$.

Entrada: $a(z) = a[0..2n-1]$.
Saída: $c(z) = a(z) \mod f(z)$.

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

Otimização para redução modular

Algoritmo 2 Otimização proposta para redução modular rápida.

Entrada: $a(z) = a[0..2n-1]$.
Saída: $c(z) = a(z) \mod f(z)$.

1: $r_b \leftarrow 0,\ r_c \leftarrow 0$
2: $i \leftarrow 21, j \leftarrow 40$
3: **while** $i > 3$ **do**
4: $\quad R(r_b, r_c, r_a, c[j]),\ c[i] \leftarrow c[i] \oplus r_b$
5: $\quad R(r_c, r_a, r_b, c[j-1]),\ c[i-1] \leftarrow c[i-1] \oplus r_c$
6: $\quad R(r_a, r_b, r_c, c[j-2]),\ c[i-2] \leftarrow c[i-2] \oplus r_a$
7: $\quad i \leftarrow i-3, j \leftarrow j-3$
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: **return** c

Análise da redução modular em $\mathbb{F}_{2^m}$

Tabela: Custo em instruções dos algoritmos de redução modular.

	Instruções executadas	
Algoritmo	Leituras	Escritas
Original	88	66
Otimização	42	22

Inversão em $\mathbb{F}_{2^m}$

Duas alternativas:

- Algoritmo Estendido de Euclides;
- Modificação do Algoritmo de Quase Inverso.

Segundo algoritmo executa apenas deslocamentos de **1 bit**!

Aritmética na curva elíptica

A multiplicação de ponto foi implementada com os algoritmos:

- 4-TNAF em curvas de Koblitz;
- Método López-Dahab em curvas binárias genéricas.

Implementação

Material:

- GCC 4.1.2 para ATMega128;
- MIRACL;
- AVR Simulator 4.14.

Linguagens:

- C;
- *Assembly*.

Curvas padronizadas:

- Curva de Koblitz `sect163k1`;
- Curva binária genérica `sect163r2`.

Resultados

Tabela: Custo dos algoritmos de aritmética em $\mathbb{F}_{2^{163}}$.

	C	*Assembly*	Ganho
Algoritmo	Ciclos	Ciclos	%
Quadrado	725	456	37
Mult. LD com reg.	13838	5433	60
Mult. LD (variante)	9752	9120	6
Mult. Karat+LD com reg.	12246	6968	43
Redução Modular	621	609	2
Inversão	365658	231258	36

Resultados

Tabela: Custo da multiplicação de ponto.

	C	*Assembly*	Ganho
Algoritmo	Tempo (s)	Tempo (s)	%
4-TNAF na curva `sect163k1`	0.95	0.69	27
LD na curva `sect163k1`	1.30	0.83	36
LD na curva `sect163r2`	1.60	0.98	39

Comparação - Eficiência

Tabela: Comparação entre implementações distintas. Os tempos são fornecidos em quantidades de ciclos (c) ou segundos (s).

	Proposta				**TinyECCK**
	Linguagem C		*Assembly*		C
Algoritmo	Tempo	Ganho	Tempo	Ganho	Tempo
Quadrado	725 c	12%	456 c	45%	825 c
Multiplicação	9752 c	51%	5433 c	72%	19670 c
Redução	621 c	67%	609 c	68%	1904 c
Inversão	365658 c	32%	231258 c	57%	539132 c
4-TNAF	0.95 s	17%	0.69 s	39%	1.14 s
LD `sect163k1`	1.30 s	-12%	0.83 s	27%	–
LD `sect163r2`	1.60 s	-29%	0.98 s	14%	–

Comparação - Memória

Tabela: Custo em *bytes* de memória para implementações distintas.

	Memória ROM	Memória RAM
4-TNAF - versão C	26668	48
4-TNAF - versão C+*Assembly*	32392	624
LD - versão C	10714	16
LD - versão C+*Assembly*	16438	528
TinyECCK	5592	618

Novo estado-da-arte para implementação de ECC em sensores:

- Implementação eficiente de aritmética no corpo $\mathbb{F}_{2^{163}}$:
  - Implementações mais eficientes de quadrado, multiplicação e redução modular já publicadas para a plataforma;
- Implementação eficiente de criptografia de curvas elípticas:
  - Multiplicação de ponto 39% mais rápida que a melhor implementação binária;
  - Multiplicação de ponto 7% mais rápida que a melhor implementação prima.

Trabalho futuro: implementação de emparelhamentos!

Corpo	Trabalho	Tempo de execução (s)
Binário	[Malan et al. 2004]	34
	[Yan and Shi 2006]	13.9
	[Eberle et al. 2005]	4.14
	[Szczechowiak et al. 2008]	2.16
	[Seo et al. 2008]	1.14
	Proposta	0.69
Primo	[Wang and Li 2006]	1.35
	[Szczechowiak et al. 2008]	1.27
	[Gura et al. 2004]	0.81
	[Uhsadel et al. 2007]	0.76
	[GrobSchadl 2006]	0.745