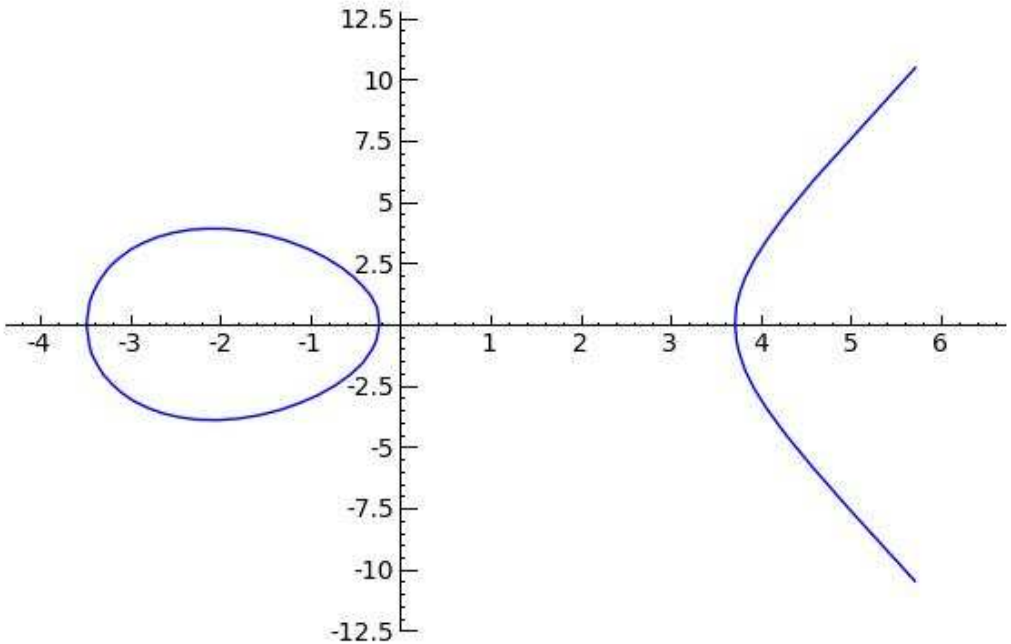
Breaking Elliptic Curves Cryptosystems using Reconfigurable Hardware

Junfeng Fan[†], Daniel V. Bailey[‡*], Lejla Batina[†◁], Tim Güneysu[‡], Christof Paar[‡] and Ingrid Verbauwhede[†]

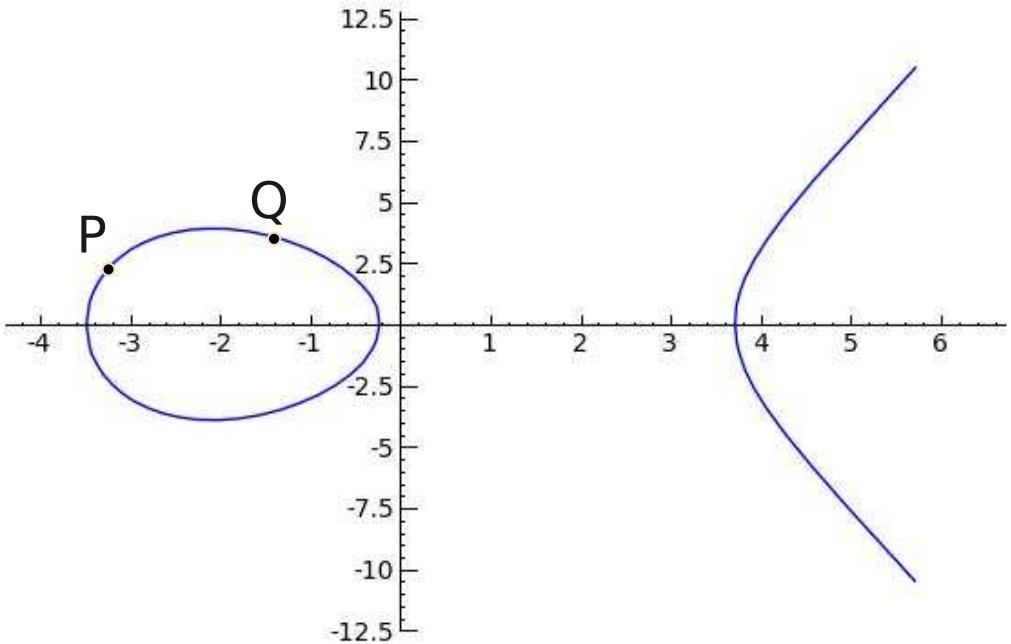
[†] ESAT/SCD-COSIC, Katholieke Universiteit Leuven and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{*Firstname.Lastname*}*@esat.kuleuven.be*
[‡] Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
{*gueneysu, cpaar*}*@crypto.rub.de*
[*] RSA, the Security Division of EMC, USA
dbailey@rsa.com
[◁] Radboud University Nijmegen, Netherlands

August 31, 2010

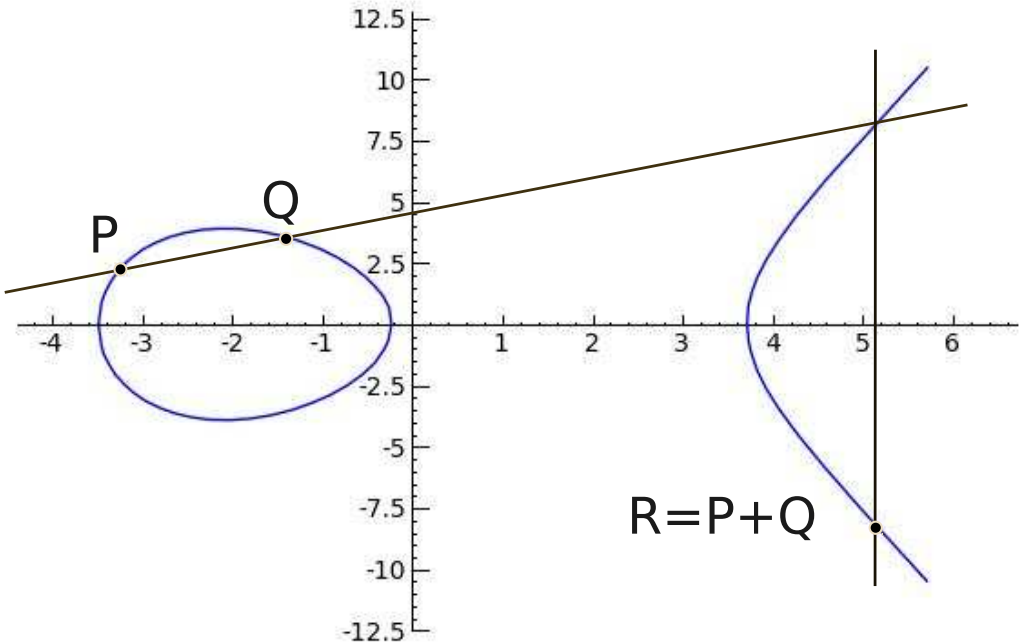
Elliptic Curve Cryptography



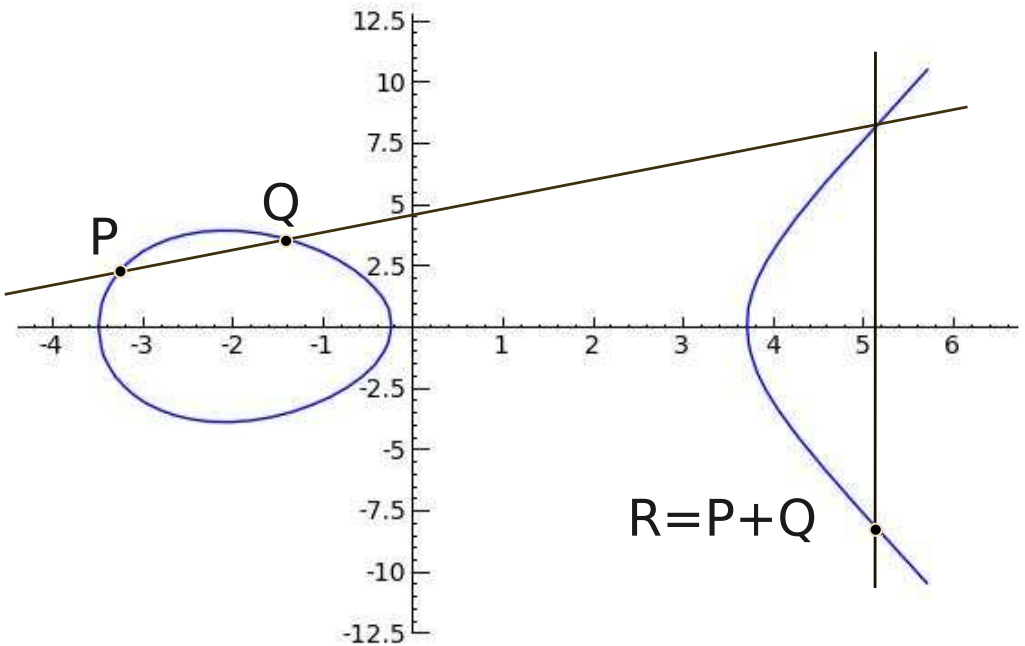
Elliptic Curve Cryptography



Elliptic Curve Cryptography



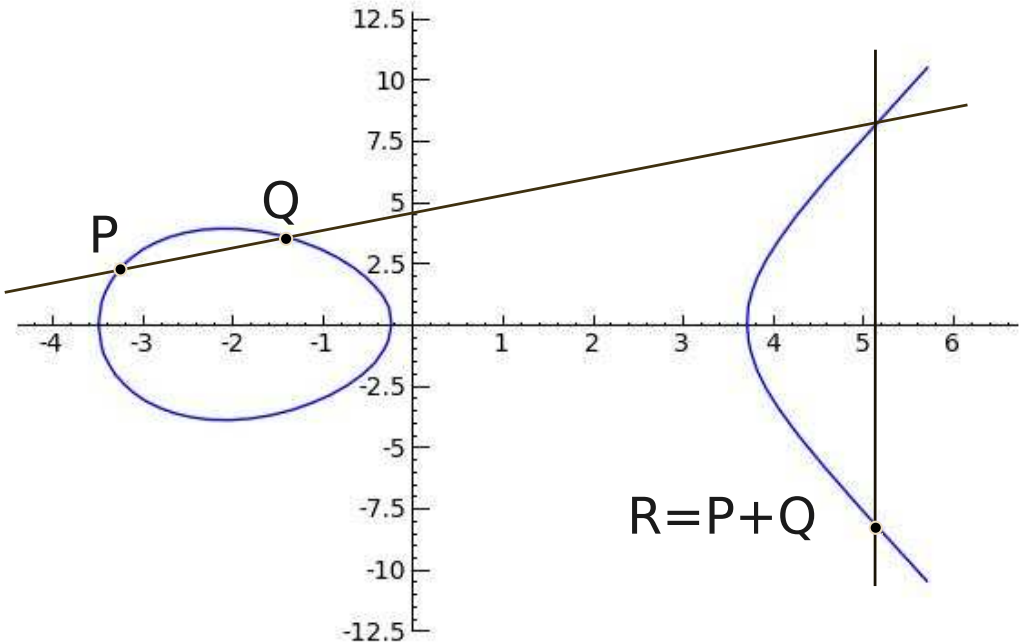
Elliptic Curve Cryptography



Point Multiplication:

$$Q \leftarrow [k]P$$
$$= P + P + \cdots + P.$$

Elliptic Curve Cryptography



Point Multiplication:

$Q \leftarrow [k]P$

$= P + P + \cdots + P.$

Elliptic Curve Discrete Logarithm Problem (ECDLP):

Given Q and P, find k such that

$Q = [k]P$.

Certicom challenge

Certicom challenge

- Lunched by Certicom in 1997

Certicom challenge

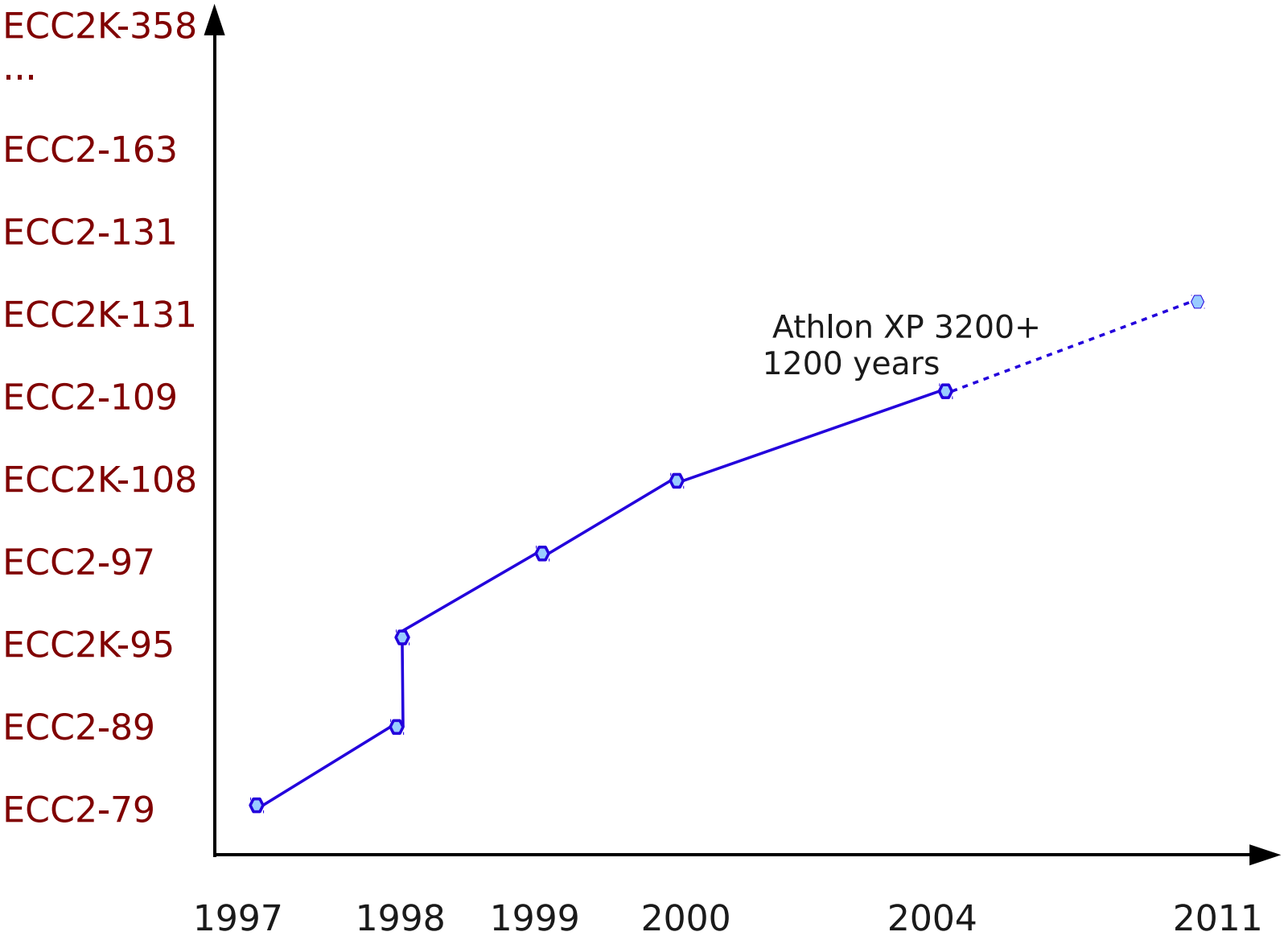
- Lunched by Certicom in 1997
- *"to increase industry understanding and appreciation for the difficulty of the elliptic curve discrete logarithm problem..."*

Certicom challenge

- Lunched by Certicom in 1997
- *"to increase industry understanding and appreciation for the difficulty of the elliptic curve discrete logarithm problem..."*

ECC on GF(2^m)	ECC on GF(p)
ECC2-79	ECCp-79
ECC2-89	ECCp-89
ECC2K-95	ECCp-97
ECC2-97	ECCp-109
ECC2K-108	ECCp-131
ECC2-109	ECCp-163
ECC2K-130	ECCp-191
ECC2-131	ECCp-239
ECC2-163	ECCp-359
ECC2K-163	
...	
ECC2K-358	

Certicom challenge



ECC2K-130

- $E : y^2 + xy = x^3 + 1$
- $F_{2^{131}} : f(w) = w^{131} + w^{13} + w^2 + w + 1$

ECC2K-130

- $E : y^2 + xy = x^3 + 1$
- $F_{2^{131}}$: $f(w) = w^{131} + w^{13} + w^2 + w + 1$
- Point P(x_P, y_P):
  x_P = 05 1C99BFA6 F18DE467 C80C23B9 8C7994AA
  y_P = 04 2EA2D112 ECEC71FC F7E000D7 EFC978BD

ECC2K-130

- E : $y^2 + xy = x^3 + 1$
- $F_{2^{131}}$: $f(w) = w^{131} + w^{13} + w^2 + w + 1$
- Point P(x_P, y_P):
  x_P = 05 1C99BFA6 F18DE467 C80C23B9 8C7994AA
  y_P = 04 2EA2D112 ECEC71FC F7E000D7 EFC978BD
- Point Q(x_Q, y_Q):
  x_Q = 06 02339C5D B0E9C694 AC890852 8C51C440
  y_Q = 04 F7B99169 FA1A0F27 37813742 B1588CB8

ECC2K-130

- E : $y^2 + xy = x^3 + 1$
- $F_{2^{131}}$: $f(w) = w^{131} + w^{13} + w^2 + w + 1$
- Point P(x_P, y_P):
  x_P = 05 1C99BFA6 F18DE467 C80C23B9 8C7994AA
  y_P = 04 2EA2D112 ECEC71FC F7E000D7 EFC978BD
- Point Q(x_Q, y_Q):
  x_Q = 06 02339C5D B0E9C694 AC890852 8C51C440
  y_Q = 04 F7B99169 FA1A0F27 37813742 B1588CB8
- **The challenge** : find k such that Q = [k]P.

The Ev1l Project

- Target : solve ECC2K-130
- Methods: Parallelized Pollard rho
- Platforms:
  - General Purpose CPU (Compute clusters)
  - Graphics card clusters
  - Playstation clusters
  - FPGA clusters
  - ASICs (Estimation)
- Status:
  - Globally distributed (Netherlands, Belgium, Ireland, USA, Taiwan, Switzerland, ..)
  - Progress report: http://www.ecc-challenge.info/

Parallelized Pollard rho

Pollard rho

finding (c_i, d_i) and (c_j, d_j) such that
$[c_i]P + [d_i]Q = [c_j]P + [d_j]Q$
$[(c_i - c_j)]P = [(d_j - d_i)]Q$
since $Q \equiv [k]P$,
$k \equiv (c_i - c_j)(d_j - d_i)^{-1} \bmod n$.

Parallelized Pollard rho

Pollard rho

finding (c_i, d_i) and (c_j, d_j) such that
$[c_i]P + [d_i]Q = [c_j]P + [d_j]Q$
$[(c_i - c_j)]P = [(d_j - d_i)]Q$
since $Q \equiv [k]P$,
$k \equiv (c_i - c_j)(d_j - d_i)^{-1} \bmod n$.

Random walk

- Start with $R_0 = [c_0]P + [d_0]Q$
- Iteration function
  $\{c_{i+1}, d_{i+1}, R_{i+1}\} \leftarrow f(c_i, d_i, R_i)$
  such that $R_i = [c_i]P + [d_i]Q$

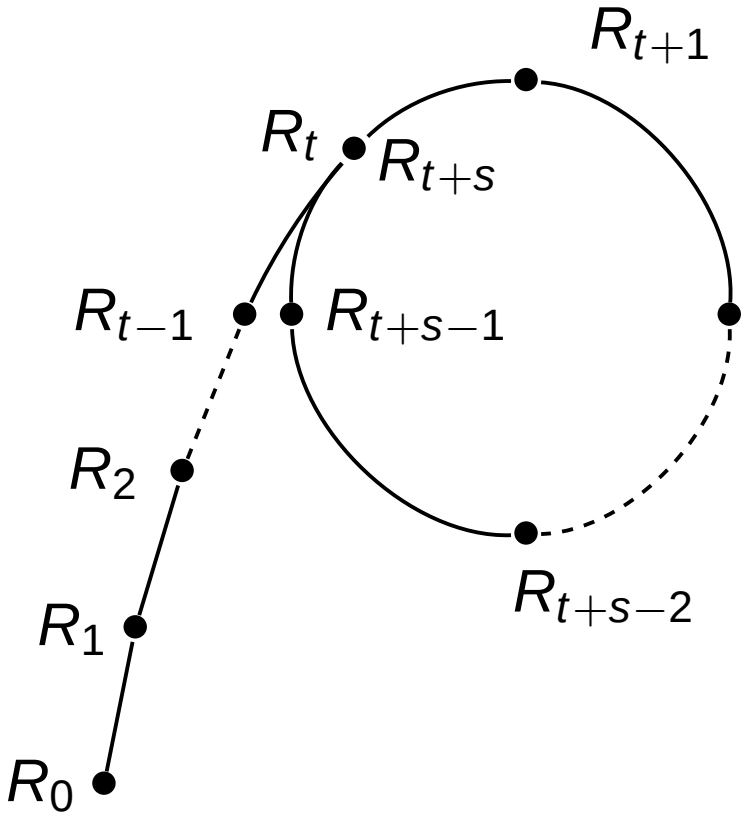
Parallelized Pollard rho

Pollard rho

finding (c_i, d_i) and (c_j, d_j) such that
$[c_i]P + [d_i]Q = [c_j]P + [d_j]Q$
$[(c_i - c_j)]P = [(d_j - d_i)]Q$
since $Q \equiv [k]P$,
$k \equiv (c_i - c_j)(d_j - d_i)^{-1} \bmod n$.

Random walk

- Start with $R_0 = [c_0]P + [d_0]Q$
- Iteration function
  $\{c_{i+1}, d_{i+1}, R_{i+1}\} \leftarrow f(c_i, d_i, R_i)$
  such that $R_i = [c_i]P + [d_i]Q$



Iteration function : $R_{i+1} \leftarrow f(R_i)$

$R_{i+1} \leftarrow \sigma^j(R_i) \oplus R_i$

Iteration function : $R_{i+1} \leftarrow f(R_i)$

$R_{i+1} \leftarrow \sigma^j(R_i) \oplus R_i$

- $j = ((HW(x_R)/2) \bmod 8) + 3$
- $\sigma^j(R) = (x_R^{2^j}, y_R^{2^j})$

Iteration function : $R_{i+1} \leftarrow f(R_i)$

$R_{i+1} \leftarrow \sigma^j(R_i) \oplus R_i$

- $j = ((HW(x_R)/2) \bmod 8) + 3$
- $\sigma^j(R) = (x_R^{2^j}, y_R^{2^j})$

Complexity of one step:

- Hamming Weight

Iteration function : $R_{i+1} \leftarrow f(R_i)$

$R_{i+1} \leftarrow \sigma^j(R_i) \oplus R_i$

- $j = ((HW(x_R)/2) \bmod 8) + 3$
- $\sigma^j(R) = (x_R^{2^j}, y_R^{2^j})$

Complexity of one step:

- Hamming Weight
- 2 m-squaring (2 m-**S**)

Iteration function : $R_{i+1} \leftarrow f(R_i)$

$R_{i+1} \leftarrow \sigma^j(R_i) \oplus R_i$

- $j = ((HW(x_R)/2) \bmod 8) + 3$
- $\sigma^j(R) = (x_R^{2^j}, y_R^{2^j})$

Complexity of one step:

- Hamming Weight
- 2 m-squaring (2 m-**S**)
- one point addition (PA)

Iteration function : $R_{i+1} \leftarrow f(R_i)$

$R_{i+1} \leftarrow \sigma^j(R_i) \oplus R_i$

- $j = ((HW(x_R)/2) \bmod 8) + 3$
- $\sigma^j(R) = (x_R^{2^j}, y_R^{2^j})$

Complexity of one step:

- Hamming Weight
- 2 m-squaring (2 m-**S**)
- one point addition (PA)
  - 2 Multiplications (2**M**)
  - 1 Inversion (1**I**)

Iteration function : $R_{i+1} \leftarrow f(R_i)$

$R_{i+1} \leftarrow \sigma^j(R_i) \oplus R_i$

- $j = ((HW(x_R)/2) \bmod 8) + 3$
- $\sigma^j(R) = (x_R^{2^j}, y_R^{2^j})$

Complexity of one step:

- Hamming Weight
- 2 m-squaring (2 m-**S**)
- one point addition (PA)
  - 2 Multiplications (2**M**)
  - 1 Inversion (1**I**)

Complexity of the attack: $2^{60.9}$ **iterations** expected!

Iteration function : $R_{i+1} \leftarrow f(R_i)$

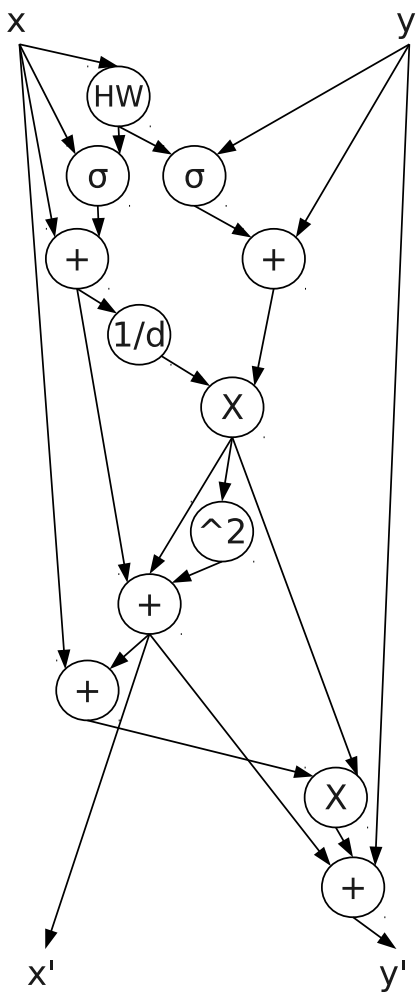
$R_{i+1} \leftarrow \sigma^j(R_i) \oplus R_i$

- $j = ((HW(x_R)/2) \bmod 8) + 3$
- $\sigma^j(R) = (x_R^{2^j}, y_R^{2^j})$

Complexity of one step:

- Hamming Weight
- 2 m-squaring (2 m-**S**)
- one point addition (PA)
  - 2 Multiplications (2**M**)
  - 1 Inversion (1**I**)

Complexity of the attack: $2^{60.9}$ **iterations** expected!



$j=(HW(x)/2 \bmod 8) +3$

$\sigma^j(x)$, $\sigma^j(y)$

$dx=x+\sigma^j(x)$,
$dy=y+\sigma^j(y)$

$s=1/dx$

$\lambda=s*dy$

λ^2

$x'=dx+\lambda+\lambda^2$

$x'+x$

$(x'+x)*\lambda$

$y'=y+x'+(x'+x)*\lambda$

FPGA platform : COPACOBANA-3

FPGA platform : COPACOBANA-3

- Xilinx Spartan-3 FPGA (XC3S5000)
- 128 FPGA units in total (16 x 8)
- 33,280 slices and 104 BRAMs available on one FPGA

Design decisions

- **Target:** Maximizing the throughput per FPGA unit (iterations per second per FPGA)

Design decisions

- **Target:** Maximizing the throughput per FPGA unit (iterations per second per FPGA)
- Representation
  - Polynomial basis:
    P $=\{ 1, w, w^2, \cdots, w^{130}\}$.
  - Type II normal basis:
    N $=\{ \gamma + \gamma^{-1}, \gamma^2 + \gamma^{-2}, \gamma^{2^2} + \gamma^{-2^2}, \cdots, \gamma^{2^{130}} + \gamma^{-2^{130}}\}$.

Design decisions

- **Target:** Maximizing the throughput per FPGA unit (iterations per second per FPGA)
- Representation
  - Polynomial basis:
    P $=\{1, w, w^2, \cdots, w^{130}\}$.
  - Type II normal basis:
    N $=\{\gamma + \gamma^{-1}, \gamma^2 + \gamma^{-2}, \gamma^{2^2} + \gamma^{-2^2}, \cdots, \gamma^{2^{130}} + \gamma^{-2^{130}}\}$.
- Multiplication algorithm
- Inversion algorithm
  - Extended Euclidean Algorithm (EEA)
  - Itoh-Tsujii (Based on Fermat Little Theorem (FLT))

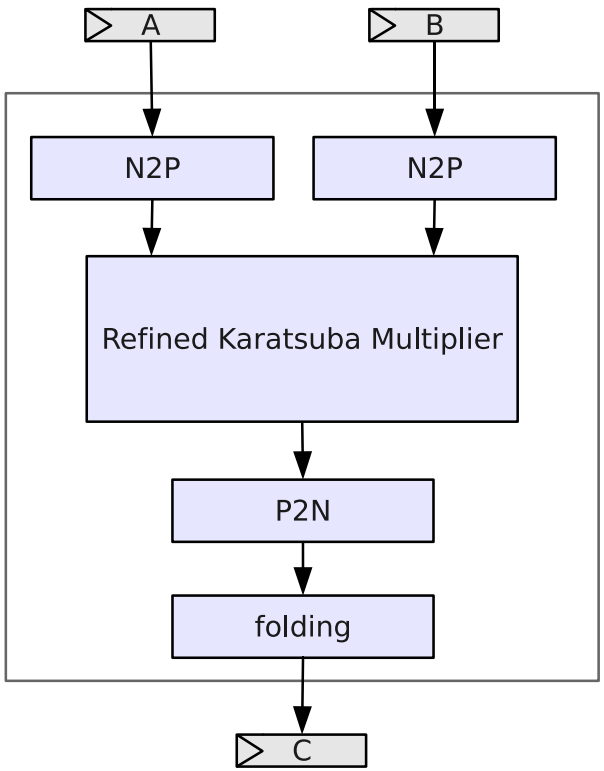
Design decisions

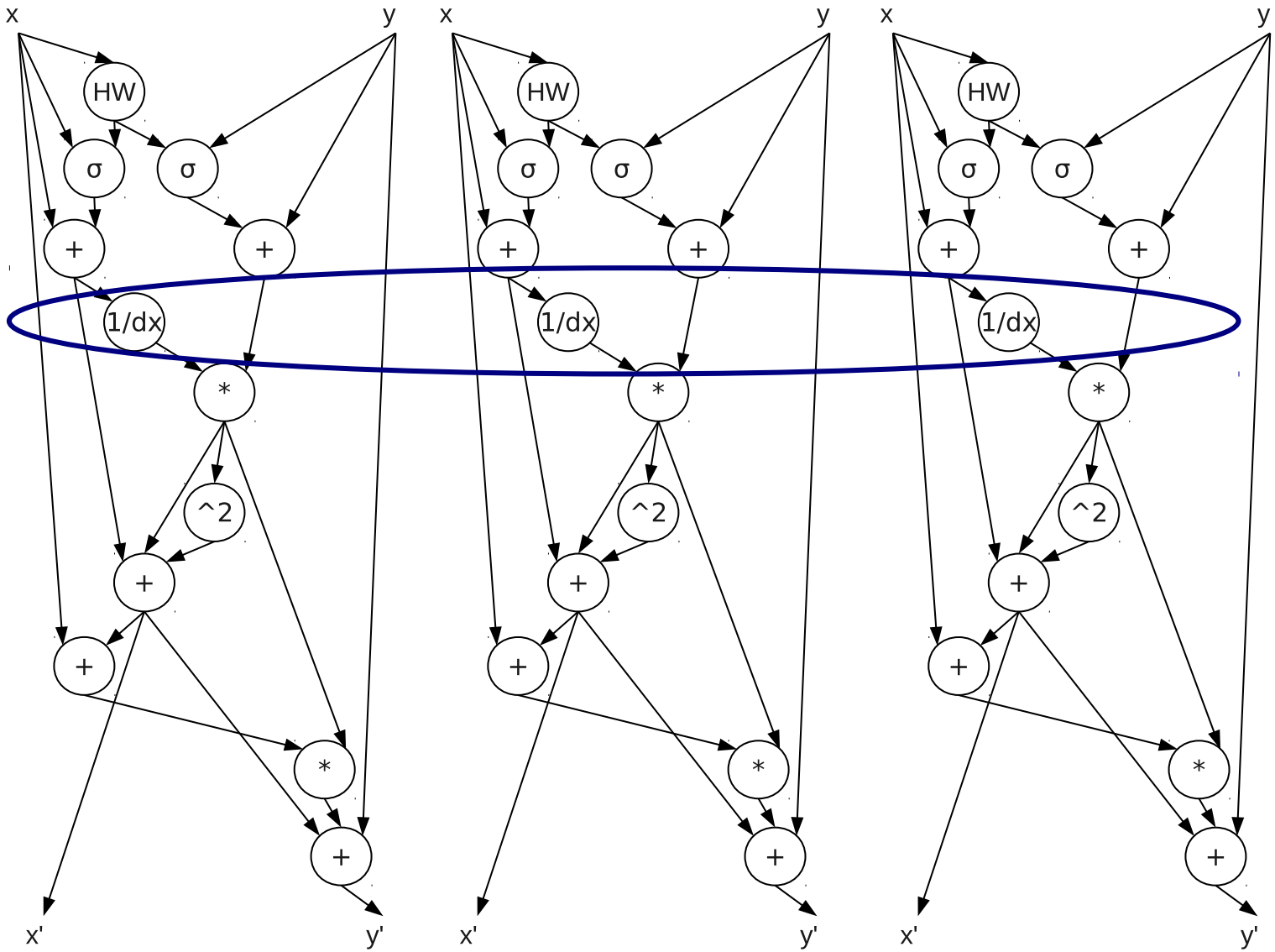
- **Target:** Maximizing the throughput per FPGA unit (iterations per second per FPGA)
- Representation
  - Polynomial basis:
    P $=\{ 1, w, w^2, \cdots, w^{130}\}$.
  - Type II normal basis:
    N $=\{ \gamma + \gamma^{-1}, \gamma^2 + \gamma^{-2}, \gamma^{2^2} + \gamma^{-2^2}, \cdots, \gamma^{2^{130}} + \gamma^{-2^{130}}\}$.
- Multiplication algorithm
- Inversion algorithm
  - Extended Euclidean Algorithm (EEA)
  - Itoh-Tsujii (Based on Fermat Little Theorem (FLT))
- Architecture
  - digit-serial
  - pipe-lined

Shokrollahi's multiplication algorithm

Fast conversion between

- permuted Normal **pN** basis
  $\{\gamma + \gamma^{-1}, \gamma^2 + \gamma^{-2}, \gamma^3 + \gamma^{-3}, \cdots, \gamma^{131} + \gamma^{-131}\}$.
- new Polynomial **nP** basis
  $\{(\gamma + \gamma^{-1}), (\gamma + \gamma^{-1})^2, \cdots, (\gamma + \gamma^{-1})^{131}\}$.




x
y
HW
σ
σ
+
+
1/dx
*
^2
+
+
*
+
x'
y'
x
y
HW
σ
σ
+
+
1/dx
*
^2
+
+
*
+
x'
y'
x
y
HW
σ
σ
+
+
1/dx
*
^2
+
+
*
+
x'
y'

Batch Inversion

Montgomery's trick for batch inversion (Batch size = 3)

Input: $\alpha_1, \alpha_2, \alpha_3$.
Output: α_1^{-1}, α_2^{-1} and α_3^{-1}.

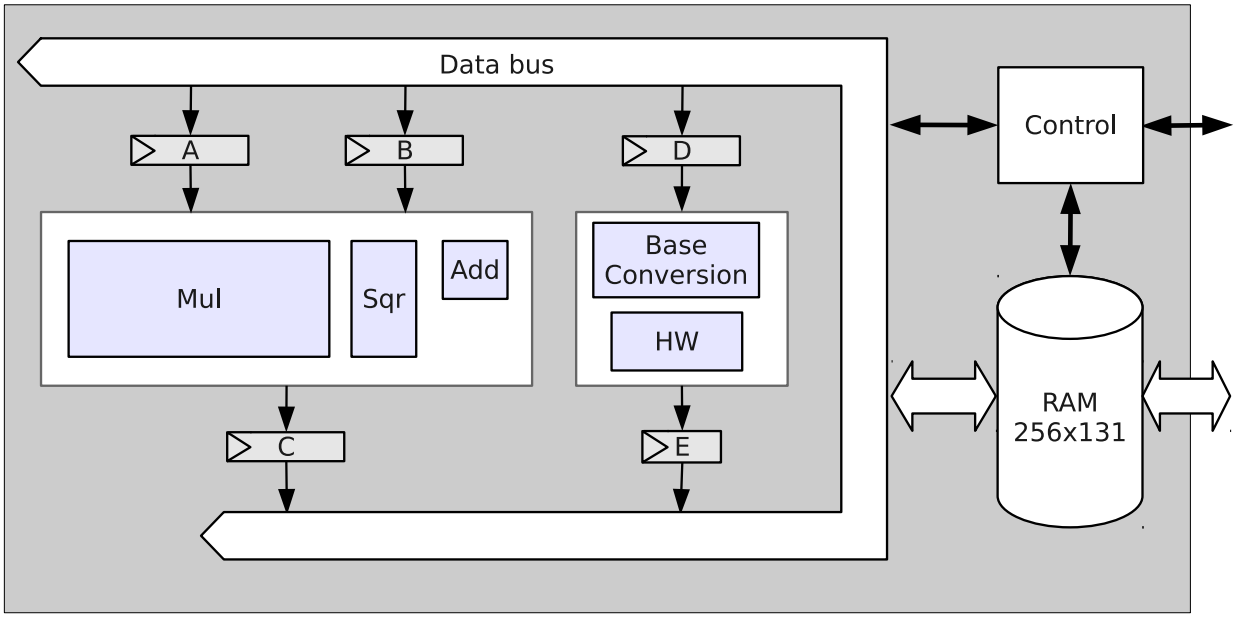
1: $d_1 \leftarrow \alpha_1$
2: $d_2 \leftarrow d_1\alpha_2$
3: $d_3 \leftarrow d_2\alpha_3$
4: $u \leftarrow d_3^{-1}$
5: $t_3 \leftarrow ud_2$, $u \leftarrow u\alpha_3$
6: $t_2 \leftarrow ud_1$, $u \leftarrow u\alpha_2$
7: $t_1 \leftarrow u$

Return: t_1, t_2, t_3.

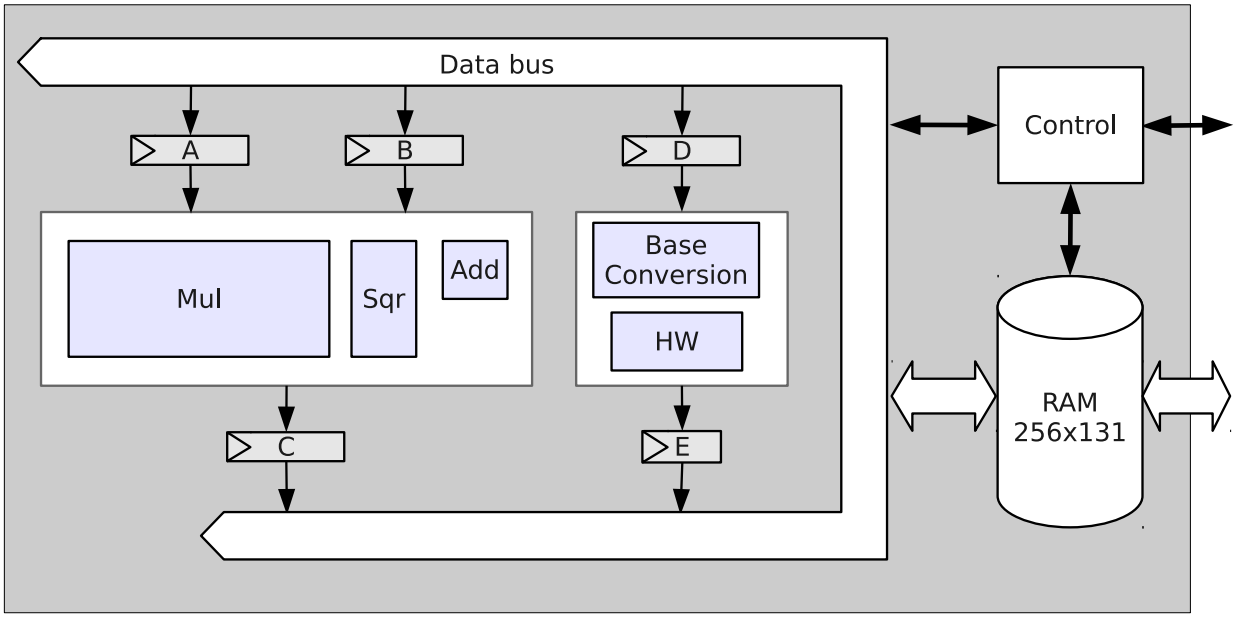
- we trade **I** for **3M + I**/n.
- One iteration: **2M + 3M + I**/n.

Polynomial basis representation, digit-serial multiplier, load/store architecture.

Polynomial basis representation, digit-serial multiplier, load/store architecture.



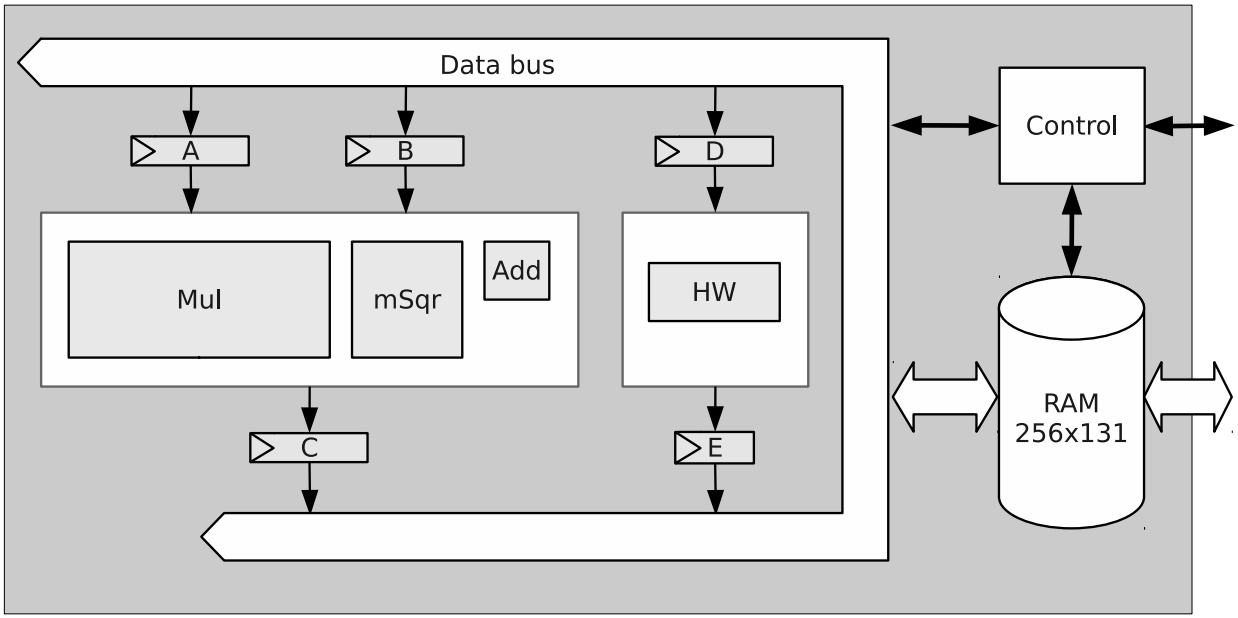
Polynomial basis representation, digit-serial multiplier, load/store architecture.



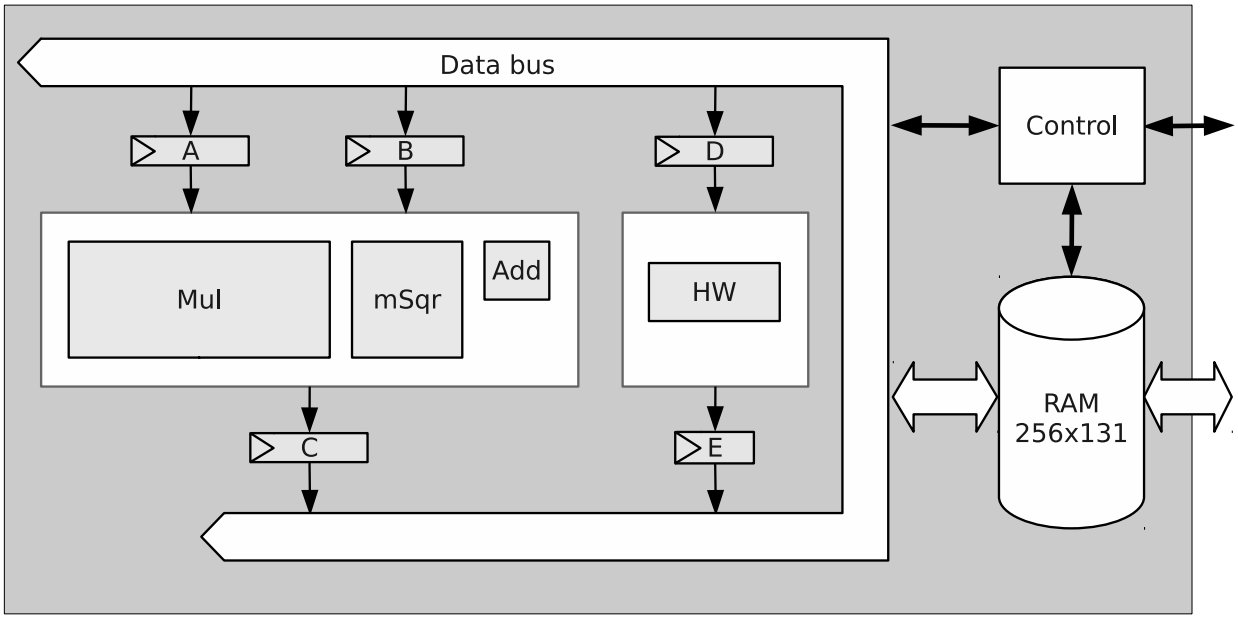
- Area : 3656 slices (Multiplier: 1468; conversion :1206)
- Frequency : 101 MHz
- 71 cycles per iteration

Normal basis representation, digit-serial multiplier (Kwon), load/store architecture

Normal basis representation, digit-serial multiplier (Kwon), load/store architecture



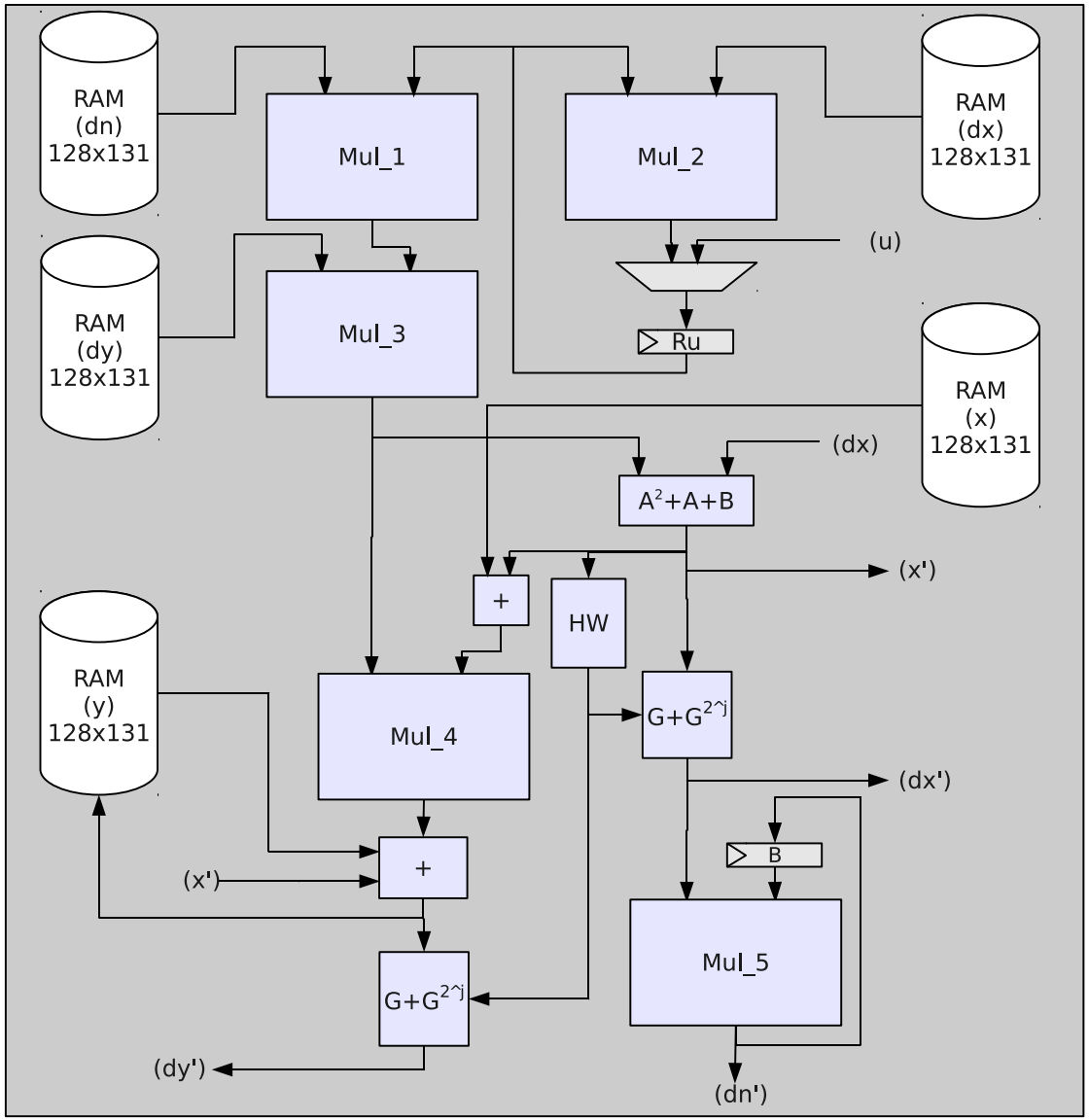
Normal basis representation, digit-serial multiplier (Kwon), load/store architecture



- Area : 2578 slices (Multiplier: 2093)
- Frequency : 125 MHz
- 81 cycles per iteration

Normal basis representation, Shorollahi multiplier (pipe-lined), Fully pipe-lined architecture

Normal basis representation, Shorollahi multiplier (pipe-lined), Fully pipe-lined architecture

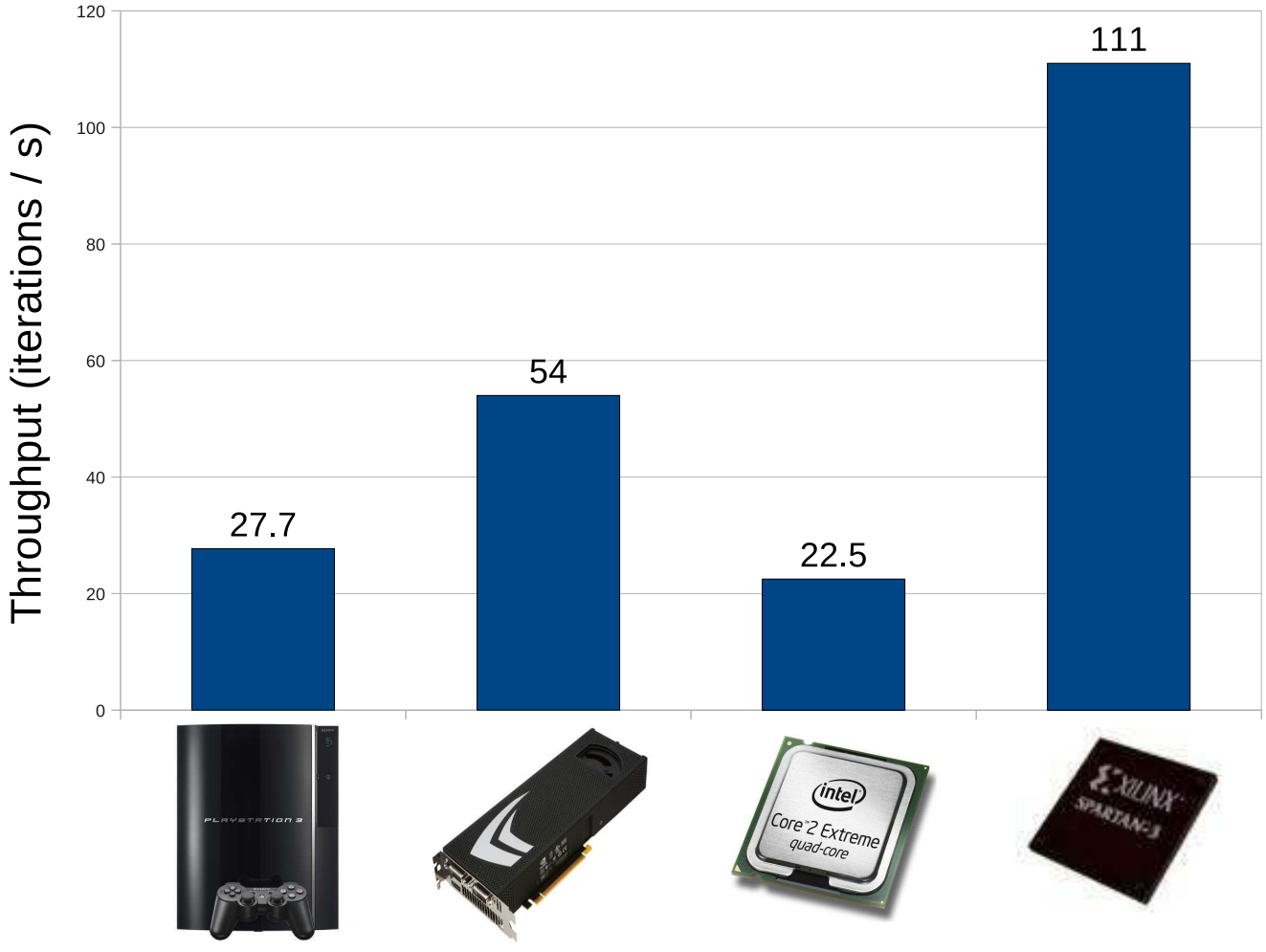


Performance of Archi-I, Archi-II and Archi-III

	Area	Freq.	Delay	Clones	Throughput
	#slice /#BRAM	[MHz]	[Cycles]		per FPGA
Archi-I: Polynomial	3,656 / 4	101	71	9	12.8×10^6
Archi-II: Type-II ONB	2,578 / 4	125	81	12	18.5×10^6
Archi-III: Shokrollahi's	26,731 / 20	111	23 *	1	111×10^6

* The average throughput is one iteration per cycle.

Comparison with CPU, Play station 3 and GPU



Effort estimation

- One FPGA finishes 111 x 2^{20} iterations per second

Effort estimation

- One FPGA finishes 111 x 2^{20} iterations per second
- One COPACOBANA-3 includes 128 FPGAs
  $\Rightarrow$ 128 x 111 x 2^{20} x 3600 x 24 x 365 = $2^{58.7}$ per year

Effort estimation

- One FPGA finishes 111 x 2^{20} iterations per second
- One COPACOBANA-3 includes 128 FPGAs
  $\Rightarrow$ 128 x 111 x 2^{20} x 3600 x 24 x 365 = $2^{58.7}$ per year
- $2^{60.9}$ iterations are expected
  $\Rightarrow$ Given 5 COPACOBANA-3 clusters, we can solve the ECC2K-130 in one year!

Conclusion

- An efficient FPGA implementation of Pollard rho attack on ECC2K-130.

Conclusion

- An efficient FPGA implementation of Pollard rho attack on ECC2K-130.
- Low-cost FPGA show good performance/cost ratio compared with CPU, Play Station 3 and GPU.

Conclusion

- An efficient FPGA implementation of Pollard rho attack on ECC2K-130.
- Low-cost FPGA show good performance/cost ratio compared with CPU, Play Station 3 and GPU.
- ECC2K-130 is practically solvable.

D. V. Bailey, L. Batina, D. J. Bernstein, P. Birkner, J. W. Bos, H. Chen, C. Cheng, G. van Damme, G. de Meulenaer, L. J. D. Perez, J. Fan, T. Güneysu, F. Gurkaynak, T. Kleinjung, T. Lange, N. Mentens, R. Niederhagen, C. Paar, F. Regazzoni, P. Schwabe, L. Uhsadel, A. Van Herrewege, and B. Yang.

Breaking ECC2K-130.

Cryptology ePrint Archive, Report 2009/541, 2009.

http://eprint.iacr.org/.

G. Meurice de Dormale, P. Bulens, and J. J. Quisquater.

Collision Search for Elliptic Curve Discrete Logarithm over GF(2^m) with FPGA.

In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007)*, pages 378–393. Springer, 2007.