KU LEUVEN

Lightweight Coprocessor for Koblitz Curves:

283-bit ECC Including Scalar Conversion with only 4300 Gates

S. Sinha Roy, **K. Järvinen**, I. Verbauwhede

KU Leuven ESAT/COSIC
Leuven, Belgium

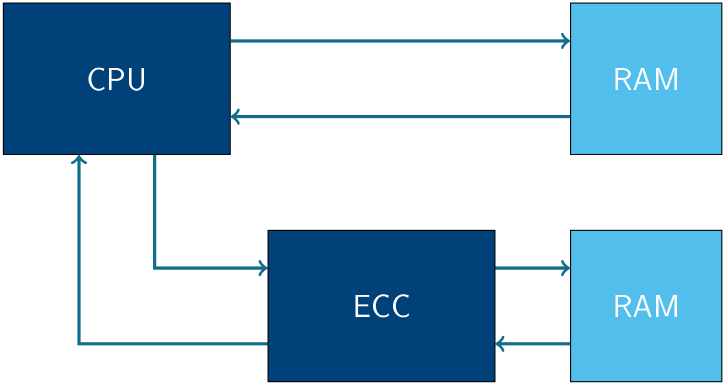
CHES 2015, Saint-Malo, France, Sept. 13–16, 2015

We present a lightweight coprocessor for the 283-bit Koblitz curve

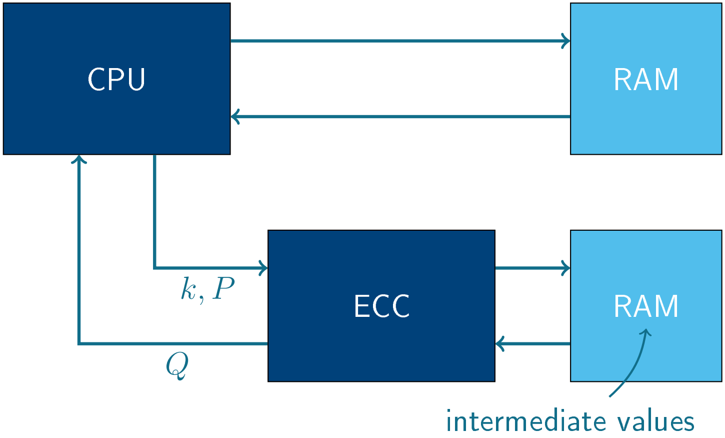
- The first lightweight implementation of a high security curve
- The first to include on-the-fly lightweight conversion
- One of the smallest ECC coprocessors
- A large set of side-channel countermeasures

Point multiplication $Q = kP$:

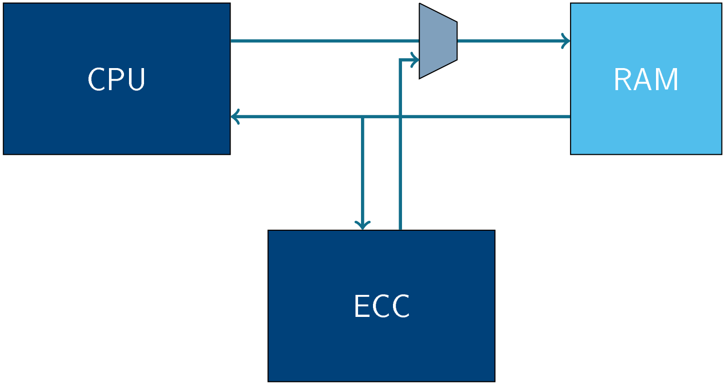
Point multiplication $Q = kP$:



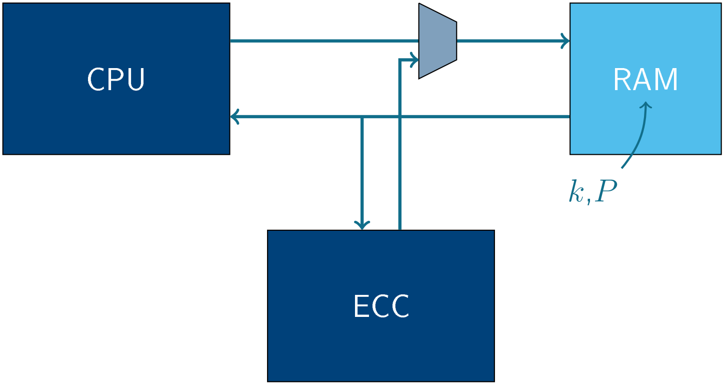
Point multiplication $Q = kP$:



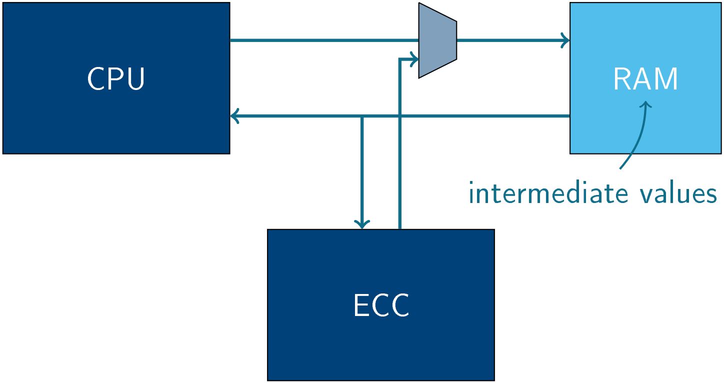
Point multiplication $Q = kP$:



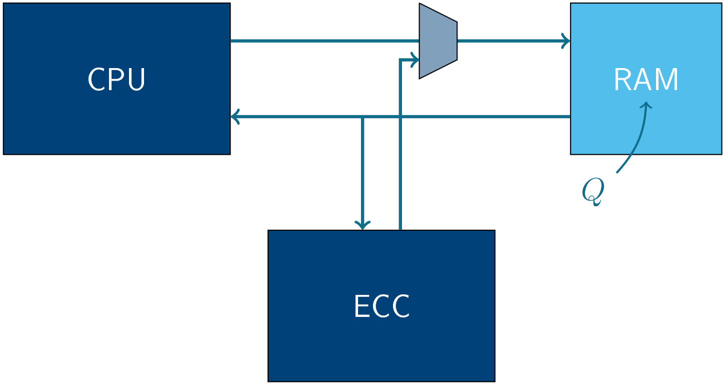
Point multiplication $Q = kP$:



Point multiplication $Q = kP$:



Point multiplication $Q = kP$:



- Binary curves which are included in many standards (e.g., NIST)

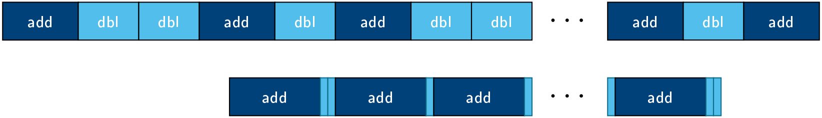
Example (Point multiplication $Q = kP$)

KU LEUVEN

Koblitz Curves

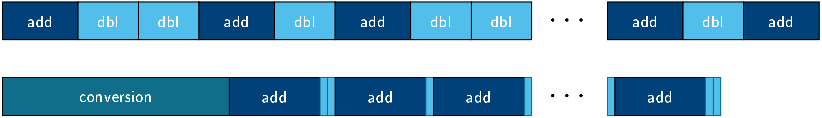
- Binary curves which are included in many standards (e.g., NIST)
- Point doublings can be replaced with cheap Frobenius maps: $\phi : (x, y) \mapsto (x^2, y^2)$

Example (Point multiplication $Q = kP$)



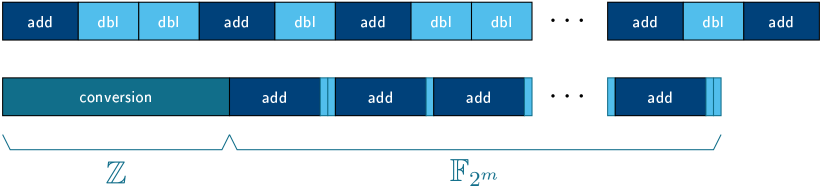
- Binary curves which are included in many standards (e.g., NIST)
- Point doublings can be replaced with cheap Frobenius maps: $\phi : (x, y) \mapsto (x^2, y^2)$
- ...but first the integer k needs to be converted to a τ-adic expansion $k = \sum_{i=0}^{\ell-1} k_i \tau^i$ where $\tau = (\mu + \sqrt{-7})/2 \in \mathbb{C}$

Example (Point multiplication $Q = kP$)



- Binary curves which are included in many standards (e.g., NIST)
- Point doublings can be replaced with cheap Frobenius maps: $\phi : (x, y) \mapsto (x^2, y^2)$
- …but first the integer k needs to be converted to a τ-adic expansion $k = \sum_{i=0}^{\ell-1} k_i \tau^i$ where $\tau = (\mu + \sqrt{-7})/2 \in \mathbb{C}$

Example (Point multiplication $Q = kP$)



Secure Lightweight Conversion

- Our conversion algorithms are based on:
  (1) the lazy reduction by Brumley and Järvinen
  (2) the zero-free expansion by Okeya, Takagi, and Vuillaume

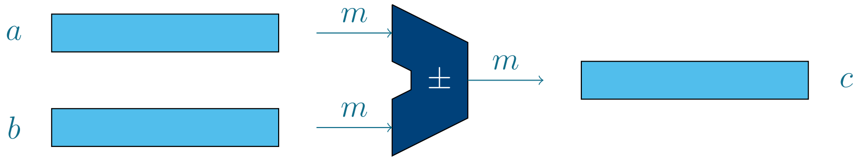
- Our conversion algorithms are based on:
  (1) the lazy reduction by Brumley and Järvinen
  (2) the zero-free expansion by Okeya, Takagi, and Vuillaume
  $\Rightarrow$ Only (multiprecision) additions and subtractions

(1): Integer k to $\rho = b_0 + b_1\tau$

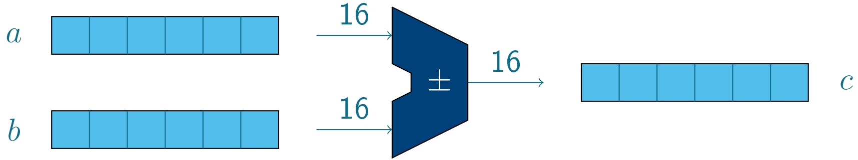
$(a_0, a_1) \leftarrow (1, 0)$, $(b_0, b_1) \leftarrow (0, 0)$,
$(d_0, d_1) \leftarrow (k, 0)$
for $i = 0$ **to** $m - 1$ **do**
  $u \leftarrow d_0 \bmod 2$
  $d_0 \leftarrow d_0 - u$
  $(b_0, b_1) \leftarrow (b_0 + u \cdot a_0, b_1 + u \cdot a_1)$
  $(d_0, d_1) \leftarrow (d_1 - d_0/2, -d_0/2)$
  $(a_0, a_1) \leftarrow (-2a_1, a_0 - a_1)$
$\rho = (b_0, b_1) \leftarrow (b_0 + d_0, b_1 + d_1)$

(2): ρ to τ-adic exp.

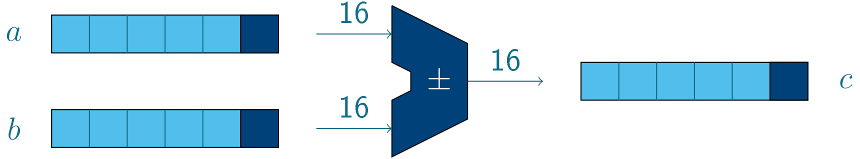
$i \leftarrow 0$
while $|b_0| \neq 1$ **or** $b_1 \neq 0$ **do**
  $u \leftarrow \Psi(b_0 + b_1\tau)$
  $b_0 \leftarrow b_0 - u$
  $(b_0, b_1) \leftarrow (b_1 - b_0/2, -b_0/2)$
  $t_i \leftarrow u$
  $i \leftarrow i + 1$
$t_i \leftarrow b_0$

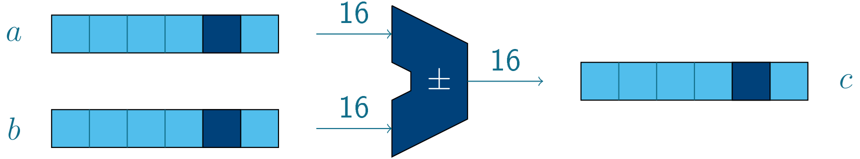


a
b
m
m
±
m
c

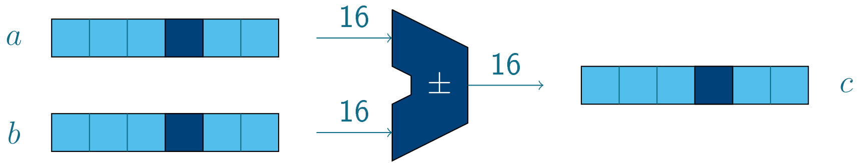


a
b
16
16
±
16
c

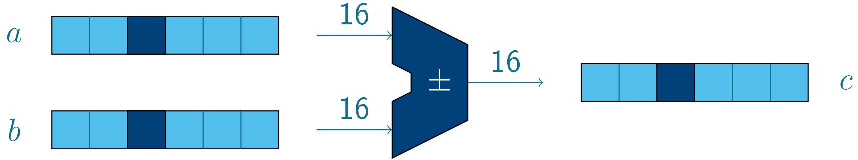

a
b
16
16
±
16
c

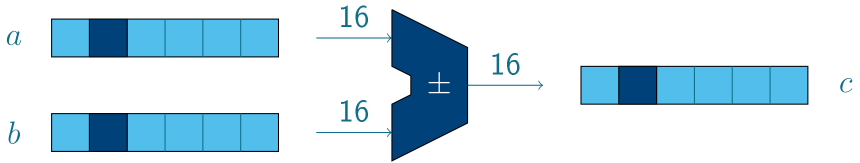

a
b
16
16
±
16
c

Modifications for Efficiency and Improved Security

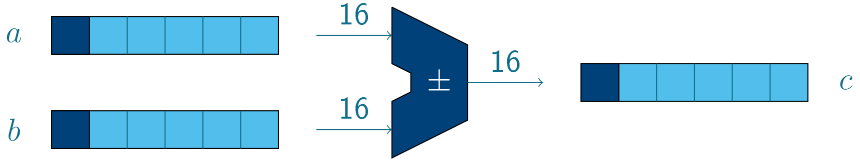


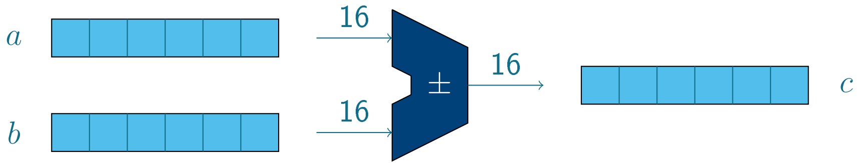
Modifications for Efficiency and Improved Security



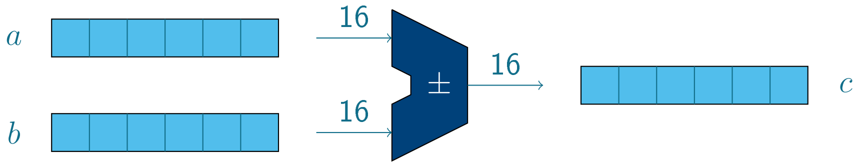


a
b
16
16
±
16
c


a
b
16
16
±
16
c



1. Negations (e.g., $-d_0/2$) take about 1/3 of cycles

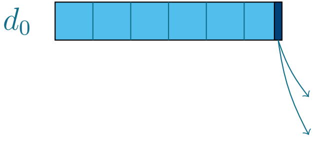


1. Negations (e.g., $-d_0/2$) take about 1/3 of cycles
   $\Rightarrow$ We use the modification $(d_0/2 - d_1, d_0/2)$ instead of $(d_1 - d_0/2, -d_0/2)$
   $\Rightarrow$ The signs will be incorrect but can be corrected

$b_i + u \cdot a_i$, where $u = d_0 \bmod 2 \in \{0, 1\}$

d_0 

$b_i + u \cdot a_i$, where $u = d_0 \bmod 2 \in \{0, 1\}$



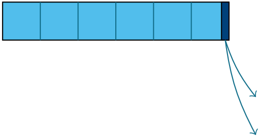
$u = 1 \Rightarrow b_0 + a_0$ and $b_1 + a_1$

$u = 0 \Rightarrow$ do nothing

Modifications for Efficiency and Improved Security (cont.)

$b_i + u \cdot a_i$, where $u = d_0 \bmod 2 \in \{0, 1\}$

d_0



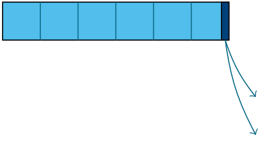
$u = 1 \Rightarrow b_0 + a_0$ and $b_1 + a_1$

$u = 0 \Rightarrow$ do nothing

Bad SPA leakage!

$b_i + u \cdot a_i$, where $u = d_0 \bmod 2 \in \{0, 1\}$

d_0



$u = 1 \Rightarrow b_0 + a_0$ and $b_1 + a_1$

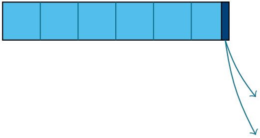
$u = 0 \Rightarrow$ do nothing

Bad SPA leakage!

2. We select $u \in \{-1, 1\}$ by using $\Psi(d_0 + d_1\tau)$

$b_i + u \cdot a_i$, where $u = d_0 \bmod 2 \in \{0, 1\}$

d_0



$u = 1 \Rightarrow b_0 + a_0$ and $b_1 + a_1$

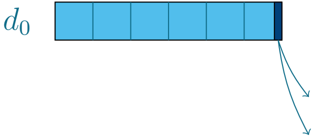
$u = 0 \Rightarrow$ do nothing

Bad SPA leakage!

2. We select $u \in \{-1, 1\}$ by using $\Psi(d_0 + d_1\tau)$
   - $u = +1 \Rightarrow b_0 + a_0$ and $b_1 + a_1$
   - $u = -1 \Rightarrow b_0 - a_0$ and $b_1 - a_1$

Modifications for Efficiency and Improved Security (cont.)

$b_i + u \cdot a_i$, where $u = d_0 \bmod 2 \in \{0, 1\}$



$u = 1 \Rightarrow b_0 + a_0$ and $b_1 + a_1$

$u = 0 \Rightarrow$ do nothing

Bad SPA leakage!

2. We select $u \in \{-1, 1\}$ by using $\Psi(d_0 + d_1\tau)$
   - $u = +1 \Rightarrow b_0 + a_0$ and $b_1 + a_1$
   - $u = -1 \Rightarrow b_0 - a_0$ and $b_1 - a_1$
   - **Similar operations $\Rightarrow$ Increased SPA resistance!**

Point Multiplication

Zero-free τ-adic expansion [Okeya et al, 2005]

A τ-adic representation that represents k with $k_i \in \{-1, 1\}$

Example

$$1\bar{1}\bar{1}1111\bar{1}111\bar{1}\bar{1}\bar{1}\ldots1\bar{1}11$$

Zero-free τ-adic expansion [Okeya et al, 2005]

A τ-adic representation that represents k with $k_i \in \{-1, 1\}$

- Combined with w-bit windows and precomputations
  - $\Rightarrow$ Fast point multiplication of only ℓ/w point additions
  - $\Rightarrow$ Constant pattern of point operations

Example

$$1\bar{1}\bar{1}1111\bar{1}111\bar{1}\bar{1}\bar{1}\ldots1\bar{1}11$$

$w = 2$:
$$P_{+1} = \phi(P) + P$$
$$P_{-1} = \phi(P) - P$$

KU LEUVEN

Zero-free τ-adic expansion [Okeya et al, 2005]

A τ-adic representation that represents k with $k_i \in \{-1, 1\}$

- Combined with w-bit windows and precomputations
  - $\Rightarrow$ Fast point multiplication of only ℓ/w point additions
  - $\Rightarrow$ Constant pattern of point operations

Example

$$\underbrace{1\bar{1}}_{+P_{-1}}\bar{1}1111\bar{1}111\bar{1}\bar{1}\bar{1}\ldots1\bar{1}11$$

$w = 2$:

$P_{+1} = \phi(P) + P$

$P_{-1} = \phi(P) - P$

Zero-free τ-adic expansion [Okeya et al, 2005]

A τ-adic representation that represents k with $k_i \in \{-1, 1\}$

- Combined with w-bit windows and precomputations
  - $\Rightarrow$ Fast point multiplication of only ℓ/w point additions
  - $\Rightarrow$ Constant pattern of point operations

Example

$$\overbrace{1\bar{1}}^{+P_{-1}}\overset{\phi^2}{\frown}\bar{1}1111\bar{1}111\bar{1}\bar{1}\bar{1}\ldots1\bar{1}11$$

$w = 2$:
$P_{+1} = \phi(P) + P$
$P_{-1} = \phi(P) - P$

Zero-free τ-adic expansion [Okeya et al, 2005]

A τ-adic representation that represents k with $k_i \in \{-1, 1\}$

- Combined with w-bit windows and precomputations
  - $\Rightarrow$ Fast point multiplication of only ℓ/w point additions
  - $\Rightarrow$ Constant pattern of point operations

Example

$$\overbrace{1\bar{1}}^{\phi^2}\bar{1}1111\bar{1}111\bar{1}\bar{1}\bar{1}\ldots1\bar{1}11$$

$1\bar{1} \Rightarrow +P_{-1}$, $\quad \bar{1}1 \Rightarrow -P_{-1}$

$w = 2$:

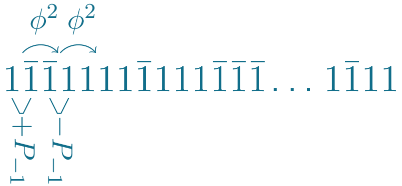
$$P_{+1} = \phi(P) + P$$
$$P_{-1} = \phi(P) - P$$

Zero-free τ-adic expansion [Okeya et al, 2005]

A τ-adic representation that represents k with $k_i \in \{-1, 1\}$

- Combined with w-bit windows and precomputations
  - $\Rightarrow$ Fast point multiplication of only ℓ/w point additions
  - $\Rightarrow$ Constant pattern of point operations

Example



$w = 2$:

$P_{+1} = \phi(P) + P$

$P_{-1} = \phi(P) - P$

KU LEUVEN

Zero-free τ-adic expansion [Okeya et al, 2005]

A τ-adic representation that represents k with $k_i \in \{-1, 1\}$

- Combined with w-bit windows and precomputations
  - $\Rightarrow$ Fast point multiplication of only ℓ/w point additions
  - $\Rightarrow$ Constant pattern of point operations

Example

ϕ^2 ϕ^2

$$1\bar{1}\bar{1}1111\bar{1}111\bar{1}\bar{1}\bar{1}\ldots1\bar{1}11$$

$+P_{-1}$ $-P_{-1}$ $+P_{+1}$

$w = 2$:

$P_{+1} = \phi(P) + P$

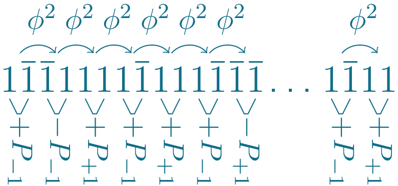
$P_{-1} = \phi(P) - P$

Zero-free τ-adic expansion [Okeya et al, 2005]

A τ-adic representation that represents k with $k_i \in \{-1, 1\}$

- Combined with w-bit windows and precomputations
  - $\Rightarrow$ Fast point multiplication of only ℓ/w point additions
  - $\Rightarrow$ Constant pattern of point operations

Example



$w = 2$:

$P_{+1} = \phi(P) + P$

$P_{-1} = \phi(P) - P$

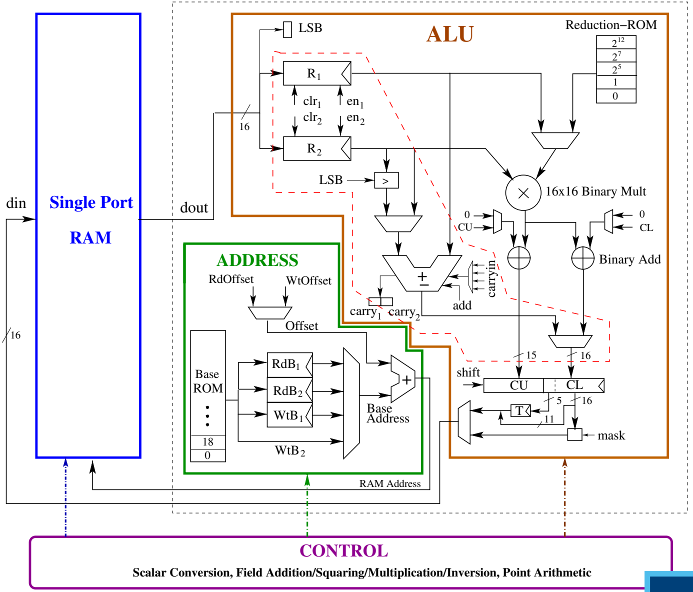
- Point additions and subtractions are computed in two phases:
  (1) To add (x, y) set $(x_p, y_p, y_m) \leftarrow (x, y, x + y)$,
  to subtract (x, y) set $(x_p, y_m, y_p) \leftarrow (x, y, x + y)$
  (2) Add (x_p, y_p, y_m)

- Point additions and subtractions are computed in two phases:
  (1) To add (x, y) set $(x_p, y_p, y_m) \leftarrow (x, y, x + y)$,
  to subtract (x, y) set $(x_p, y_m, y_p) \leftarrow (x, y, x + y)$
  (2) Add (x_p, y_p, y_m)
- The accumulator point is randomized as shown by Coron: $(X, Y, Z) = (xr, yr^2, r)$, where r is random

- Point additions and subtractions are computed in two phases:
  (1) To add (x, y) set $(x_p, y_p, y_m) \leftarrow (x, y, x + y)$,
  to subtract (x, y) set $(x_p, y_m, y_p) \leftarrow (x, y, x + y)$
  (2) Add (x_p, y_p, y_m)
- The accumulator point is randomized as shown by Coron: $(X, Y, Z) = (xr, yr^2, r)$, where r is random
- The expansion is expanded up to (almost) constant length

- Point additions and subtractions are computed in two phases:
  (1) To add (x, y) set $(x_p, y_p, y_m) \leftarrow (x, y, x + y)$,
  to subtract (x, y) set $(x_p, y_m, y_p) \leftarrow (x, y, x + y)$
  (2) Add (x_p, y_p, y_m)
- The accumulator point is randomized as shown by Coron: $(X, Y, Z) = (xr, yr^2, r)$, where r is random
- The expansion is expanded up to (almost) constant length
- The attacker can obtain only a single trace from the conversion

Architecture and Results



ALU
Reduction–ROM
2^12
2^7
2^5
1
0
LSB
R1
R2
clr1
en1
clr2
en2
16
LSB
>
16x16 Binary Mult
0
CU
0
CL
Binary Add
carryin
add
carry1
carry2
15
16
shift
CU
CL
5
16
T
11
mask
din
Single Port
RAM
dout
16
ADDRESS
RdOffset
WtOffset
Offset
Base ROM
18
0
RdB1
RdB2
WtB1
WtB2
Base Address
RAM Address
CONTROL
Scalar Conversion, Field Addition/Squaring/Multiplication/Inversion, Point Arithmetic

We synthesized the design (coprocessor, not RAM) for UMC 130 nm CMOS with Synopsys Design Compiler

- 4,323 GE
- 1,566,000 clock cycles (incl. conversion)
- 97.89 ms (@16 MHz)
- 97.70 μW (@16 MHz)
- 9.56 μJ (@16 MHz)

Results and Comparisons (Cont.)

Work	Curve	RAM	Area (GE)	Latency (cycles)	Latency (ms)	Power (μW)
Batina'06	B-163	no	9,926	95,159	190.32	<60
Bock'08	B-163	yes	12,876	–	95	93
Hein'08	B-163	yes	13,250	296,299	2,792	80.85
Kumar'06	B-163	yes	16,207	376,864	27.90	n/a
Lee'08	B-163	yes	12,506	275,816	244.08	32.42
Wegner'11	B-163	yes	8,958	286,000	2,860	32.34
Wegner'13	B-163	no	4,114	467,370	467.37	66.1
Pessl'14	P-160	yes	12,448	139,930	139.93	42.42
Azarderakhsh'14	K-163	yes	11,571	106,700	7.87	5.7
Our, est.	B-163	no	≈3,773	≈485,000	≈30.31	≈6.11
Our, est.	K-163	no	≈4,323	≈420,900	≈26.30	≈6.11
Our, est.	B-283	no	≈3,773	≈1,934,000	≈120.89	≈6.11
Our, est.	K-283	yes⋆	10,204⋆	1,566,000	97.89	>6.11
Our	**K-283**	**no**	**4,323**	**1,566,000**	**97.89**	**6.11**

⋆ Estimate for a 256×16-bit RAM, space needed for 252 16-bit words (4032 bits)

We showed that

- 283-bit curves are feasible for lightweight implementations
  - $\Rightarrow$ The price to pay comes mainly in latency and memory requirements

We showed that

- 283-bit curves are feasible for lightweight implementations
  - ⇒ The price to pay comes mainly in latency and memory requirements
- Koblitz curves are feasible for lightweight implementations
  - ⇒ Lead to savings in latency and energy consumption

We showed that

- 283-bit curves are feasible for lightweight implementations
  - ⇒ The price to pay comes mainly in latency and memory requirements
- Koblitz curves are feasible for lightweight implementations
  - ⇒ Lead to savings in latency and energy consumption
- The drop-in concept is very efficient for high security curves
  - ⇒ Area of the memory becomes less of an issue

We showed that

- 283-bit curves are feasible for lightweight implementations
  - $\Rightarrow$ The price to pay comes mainly in latency and memory requirements
- Koblitz curves are feasible for lightweight implementations
  - $\Rightarrow$ Lead to savings in latency and energy consumption
- The drop-in concept is very efficient for high security curves
  - $\Rightarrow$ Area of the memory becomes less of an issue

Future work

- Careful validation of resistance against side-channel attacks

Thank you! Questions?