Implementing Complete Formulas on Weierstrass Curves in Hardware

Pedro Maat Massolino Joost Renes Lejla Batina

[1]Radboud University, Digital Security, Nijmegen, The Netherlands
P.Massolino,j.renes,lejla@cs.ru.nl

23rd December 2016

Outline

- Elliptic curve preliminaries
- Problem of exceptional cases
- Complete addition formulas
- Parallelizing the formulas
- Hardware architecture
- Results
- Preliminary side-channel analysis
- Conclusions

Elliptic curves

$E(k)$: elliptic curve over a field k with $\mathrm{char}(k) \neq 2, 3$

Every elliptic curve can be written in short Weierstrass form

- Embedded in $\mathbb{P}^2(k)$ as $E : Y^2Z = X^3 + aXZ^2 + bZ^3$
- The point $\mathcal{O} = (0 : 1 : 0)$ is called the point at infinity
- Affine points $(x : y : 1)$ given by $y^2 = x^3 + ax + b$

- The points on E form an abelian group under point addition $\oplus$ (with neutral element $\mathcal{O}$)
- Scalar multiplication $(k, P) \mapsto [k]P$ $(k \in \mathbb{Z}, P \in E)$
- The order of E is its order as a group

Elliptic curve cryptography (ECC)

Elliptic curve discrete logarithm problem (ECDLP)

Given two points $P, Q \in E$ such that $Q \in \langle P \rangle$. Find $k \in \mathbb{Z}$ such that $Q = [k]P$.

Commonly k is a secret, Q is public

- Key exchange: ECDH
- Signatures: ECDSA, EdDSA, SchnorrQ

Weierstrass model

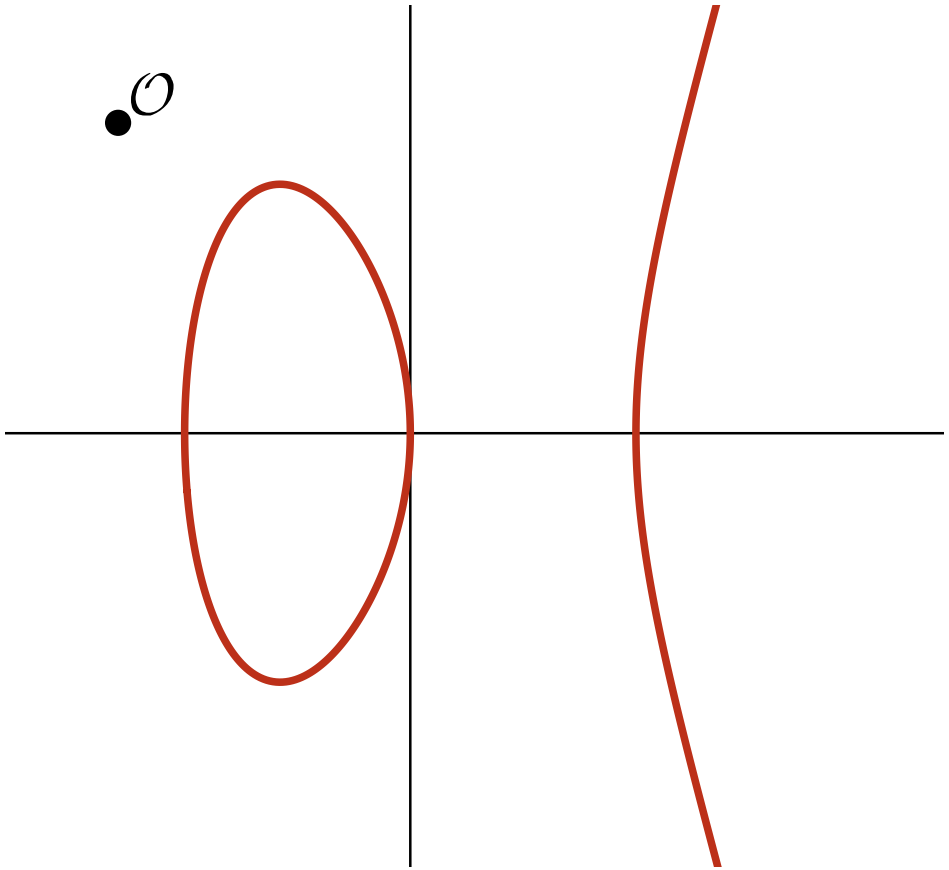


Figure: $E/\mathbb{R} : y^2 = x^3 + ax + b$

Weierstrass model over a finite field

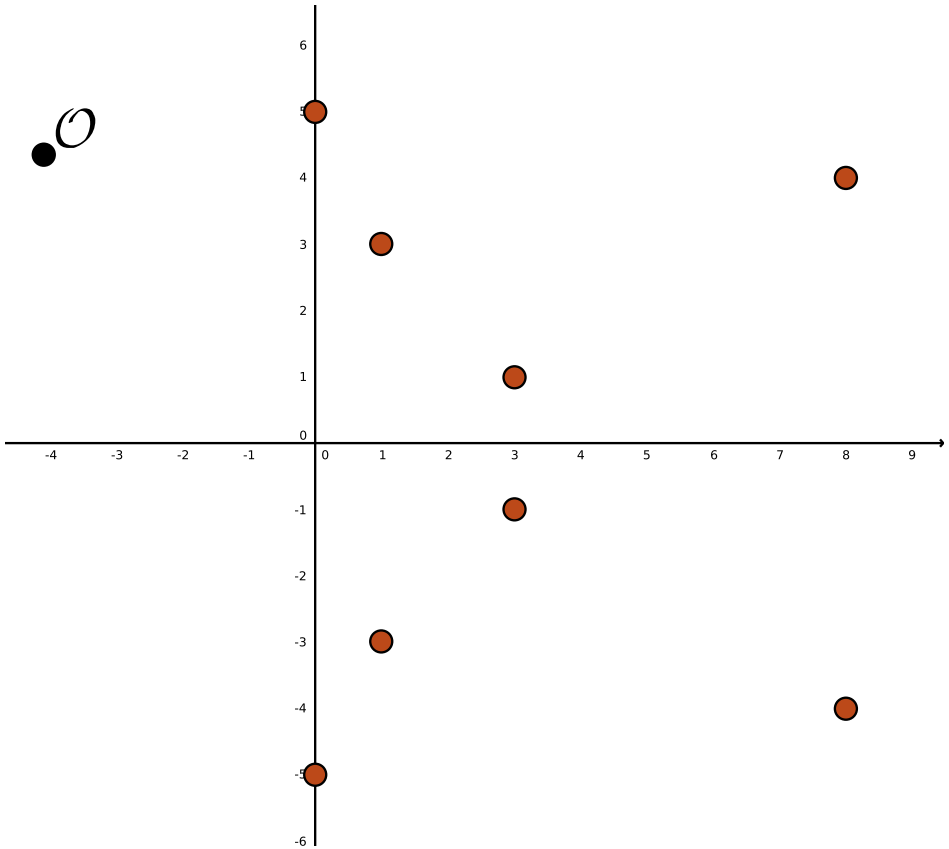


Figure: $E/\mathbb{F}_{11} : y^2 = x^3 + 5x + 3$

Chord and tangent addition

- if $P \neq \pm Q$
- if $P \neq \mathcal{O}$
- if $Q \neq \mathcal{O}$

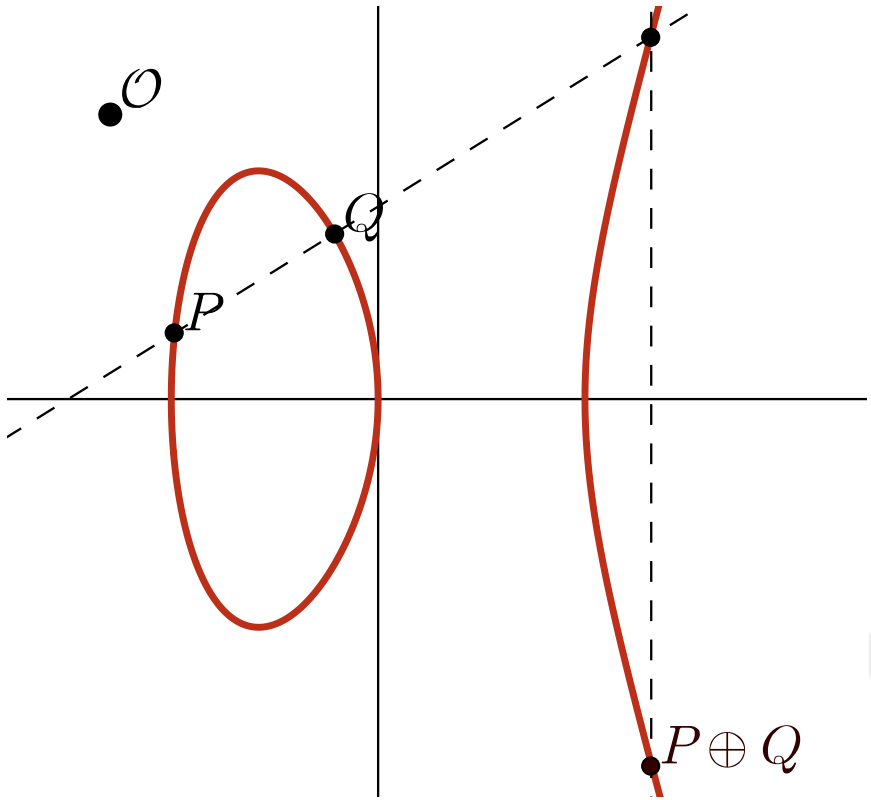


Figure: $E/\mathbb{R} : y^2 = x^3 + ax + b$

Weierstrass model doubling

- if $P \neq \mathcal{O}$

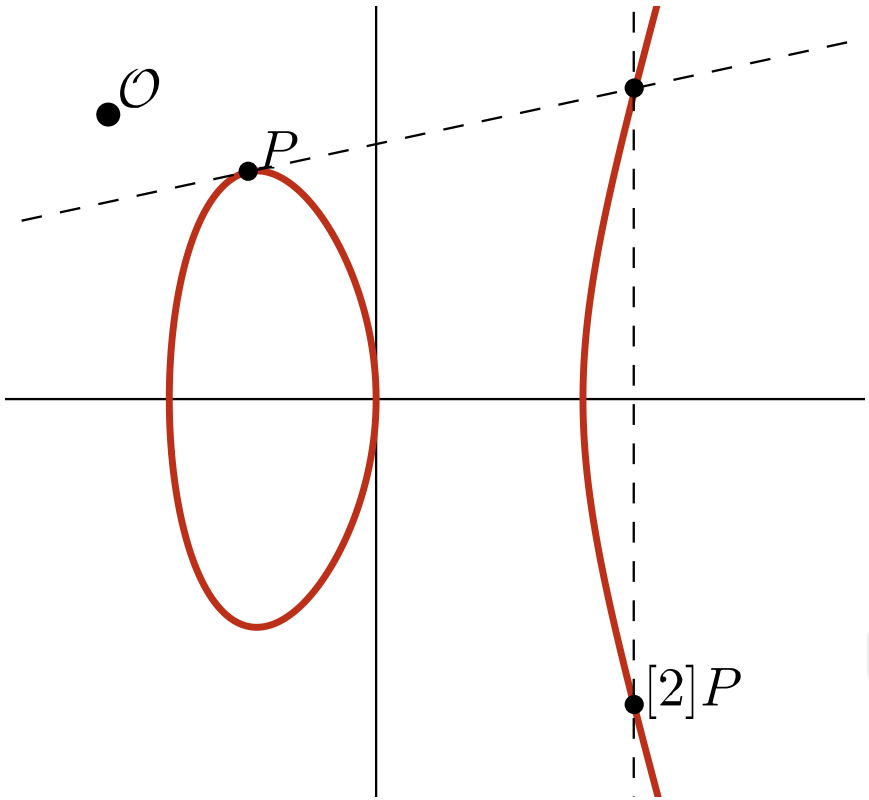


Figure: $E/\mathbb{R} : y^2 = x^3 + ax + b$

Implementation (Homogeneous addition)

Addition of $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$:
$(X_1 : Y_1 : Z_1) \oplus (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$, where:

$$X_3 = (X_2Z_1 - X_1Z_2)\Big[(Y_2Z_1 - Y_1Z_2)Z_1Z_2 - (X_2Z_1 - X_1Z_2)^3 - 2(X_2Z_1 - X_1Z_2)X_1Z_2\Big],$$

$$Y_3 = (Y_2Z_1 - Y_1Z_2)\Big[3(X_2Z_1 - X_1Z_2)X_1Z_2 - (Y_2Z_1 - Y_1Z_2)Z_1Z_2 + (X_2Z_1 - X_1Z_2)^3\Big] - (X_2Z_1 - X_1Z_2)^3Y_1Z_2,$$

$$Z_3 = (X_2Z_1 - X_1Z_2)^3Z_1Z_2.$$

But: $\left.\begin{array}{l} P = Q \\ P = \mathcal{O} \\ Q = \mathcal{O} \end{array}\right\} \Longrightarrow X_3 = Y_3 = Z_3 = 0$ (**not in $\mathbb{P}^2$!**)

Implementation (Homogeneous doubling)

Doubling of $P = (X : Y : Z)$ to compute $2P = (X_3 : Y_3 : Z_3)$:
$[2](X : Y : Z) = (X_3 : Y_3 : Z_3)$, where

$$X_3 = 2\Big[(aZ^2 + 3X^2)^2 - 8XY^2Z\Big]YZ,$$
$$Y_3 = (aZ^2 + 3X^2)\Big[12XY^2Z - (aZ^2 + 3X^2)^2\Big] - 8Y^4Z^2,$$
$$Z_3 = 8Y^3Z^3.$$

But: $P = \mathcal{O} \implies X_3 = Y_3 = Z_3 = 0$ (**not in $\mathbb{P}^2$!**)

Curve model

- Problems appear for curves in short Weierstrass form
- Can deal with the exceptions by changing the model
  - (twisted) Edwards
  - (twisted) Hessian
- Not possible for prime order curves

Prime order curves

- The example curves originally specified in the working drafts of ANSI, versions X9.62 and X9.63.
- The five NIST prime curves specified in FIPS 186-4, i.e. P-192, P-224, P-256, P-384 and P-521.
- The seven curves specified in the German Brainpool standard, where $l \in \{160, 192, 224, 256, 320, 384, 512\}$, l is the bit-length of p.
- The eight curves specified by the UK-based company Certivox, where $l \in \{160, 192, 224, 256, 288, 320, 384, 512\}$.
- The three curves specified (in addition to the above NIST prime curves) in the Certicom SEC 2 standard. This includes `secp256k1`, which is the curve used in the Bitcoin protocol.

The formulas

Complete addition formulas for odd order elliptic curves. For any two points $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$ we can compute $P + Q = (X_3 : Y_3 : Z_3)$ where

$$
\begin{aligned}
X_3 &= (X_1Y_2 + X_2Y_1)(Y_1Y_2 - a(X_1Z_2 + X_2Z_1) - 3bZ_1Z_2) \\
&\quad - (Y_1Z_2 + Y_2Z_1)(aX_1X_2 + 3b(X_1Z_2 + X_2Z_1) - a^2Z_1Z_2), \\
Y_3 &= (Y_1Y_2 + a(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2)(Y_1Y_2 - a(X_1Z_2 + X_2Z_1) - 3bZ_1Z_2) \\
&\quad + (3X_1X_2 + aZ_1Z_2)(aX_1X_2 + 3b(X_1Z_2 + X_2Z_1) - a^2Z_1Z_2), \\
Z_3 &= (Y_1Z_2 + Y_2Z_1)(Y_1Y_2 + a(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2) \\
&\quad + (X_1Y_2 + X_2Y_1)(3X_1X_2 + aZ_1Z_2).
\end{aligned}
$$

Exceptional pairs are induced by points of order 2, which by assumption only exist over extension fields.

Operation count

$$\text{any } a\text{:} \quad \begin{cases} 12\mathbf{M} + 3\mathbf{m_a} + 2\mathbf{m_{3b}} + 23\mathbf{a} & P \oplus Q \\ 11\mathbf{M} + 3\mathbf{m_a} + 2\mathbf{m_{3b}} + 17\mathbf{a} & P \oplus Q, Z_Q = 1 \\ 8\mathbf{M} + 3\mathbf{S} + 3\mathbf{m_a} + 2\mathbf{m_{3b}} + 15\mathbf{a} & [2]P \end{cases}$$

$$a = -3\text{:} \quad \begin{cases} 12\mathbf{M} + 2\mathbf{m_b} + 29\mathbf{a} & P \oplus Q \\ 11\mathbf{M} + 2\mathbf{m_b} + 23\mathbf{a} & P \oplus Q, Z_Q = 1 \\ 8\mathbf{M} + 3\mathbf{S} + 2\mathbf{m_b} + 21\mathbf{a} & [2]P \end{cases}$$

$$a = 0\text{:} \quad \begin{cases} 12\mathbf{M} + 2\mathbf{m_{3b}} + 19\mathbf{a} & P \oplus Q \\ 11\mathbf{M} + 2\mathbf{m_{3b}} + 13\mathbf{a} & P \oplus Q, Z_Q = 1 \\ 6\mathbf{M} + 2\mathbf{S} + 1\mathbf{m_{3b}} + 9\mathbf{a} & [2]P \end{cases}$$

A comparison

- **Complete addition formulas [RCB16]**:
  $12\mathbf{M} + 3\mathbf{m_a} + 2\mathbf{m_{3b}} + 23\mathbf{a}$
- **Exception-free doubling [RCB16]**:
  $8\mathbf{M} + 3\mathbf{S} + 3\mathbf{m_a} + 2\mathbf{m_{3b}} + 15\mathbf{a}$
- Bernstein and Lange attempt an addition law which works for all NIST prime curves: $26\mathbf{M} + 8\mathbf{S} + \ldots$
- Brier and Joye [BJ02] develop *unified* formulas, still with exceptions: $11\mathbf{M} + 6\mathbf{S} + \ldots$
- Bos *et al.* [BCLN15] study a complete *system* of two addition laws
- Chord-and-tangent Jacobian coordinates addition:
  $\approx 12\mathbf{M} + 4\mathbf{S} + \ldots$
- Chord-and-tangent Jacobian coordinates doubling:
  $\approx 4\mathbf{M} + 4\mathbf{S} + \ldots$

Hardware implementation: first observations

Building on top of a Mongomery modular multiplier:

- Using Montgomery representation, making additions/subtractions very fast
- No distinction between multiplications and squarings
- Multiplications by constants are cheap (if predefined)
- Can use multiple multipliers i.e. benefit from parallelism

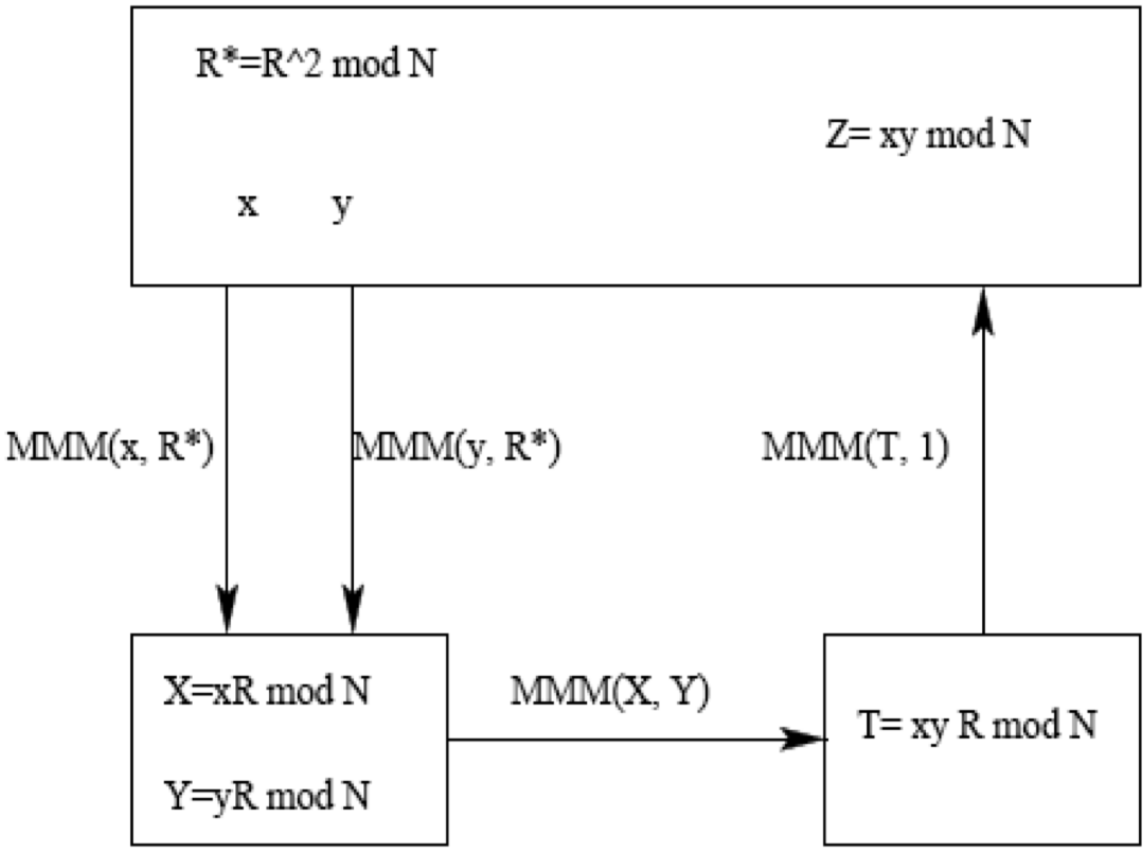
Montgomery Multiplication Method (MMM)

- invented by P. Montgomery in 1985
- a clever way to deal with trial divisions in hardware

Montgomery representation

- $x \rightarrow X = xR \mod N$ where $R = 2^r = (2^\alpha)^n > N, b = 2^\alpha$
- $MMM(X, Y) = XYR^{-1} \mod N$

Montgomery multiplication



Hardware considerations

- modular additions/subtractions can be very cheap
- comes down to looking into multiplications
- using n Montgomery multipliers
- consider $12\mathbf{M} + 3\mathbf{m_a} + 2\mathbf{m_{3b}}$ close to $17\mathbf{M}$
  - For 256 bit arithmetic, we have $1\mathbf{M} \approx 15\mathbf{a}$
  - For 512 bit arithmetic, we have $1\mathbf{M} \approx 25\mathbf{a}$

Interdependencies of multiplications

stage	result	multiplication	dependent on
0	ℓ_0	$X_1 \cdot X_2$	-
0	ℓ_1	$Y_1 \cdot Y_2$	-
0	ℓ_2	$Z_1 \cdot Z_2$	-
0	ℓ_3	$(X_1 + Y_1) \cdot (X_2 + Y_2)$	-
0	ℓ_4	$(X_1 + Z_1) \cdot (X_2 + Z_2)$	-
0	ℓ_5	$(Y_1 + Z_1) \cdot (Y_2 + Z_2)$	-
1	ℓ_6	$b_3 \cdot \ell_2$	ℓ_2
1	ℓ_7	$a \cdot \ell_2$	ℓ_2
1	ℓ_8	$a \cdot (\ell_4 - \ell_0 - \ell_2)$	ℓ_0, ℓ_2, ℓ_4
1	ℓ_9	$b_3 \cdot (\ell_4 - \ell_0 - \ell_2)$	ℓ_0, ℓ_2, ℓ_4
2	ℓ_{10}	$a \cdot (\ell_0 - \ell_7)$	ℓ_0, ℓ_7
2	ℓ_{11}	$(\ell_3 - \ell_0 - \ell_1) \cdot (\ell_1 - \ell_8 - \ell_6)$	$\ell_0, \ell_1, \ell_3, \ell_6, \ell_8$
2	ℓ_{13}	$(\ell_1 + \ell_8 + \ell_6) \cdot (\ell_1 - \ell_8 - \ell_6)$	ℓ_1, ℓ_6, ℓ_8
2	ℓ_{15}	$(\ell_5 - \ell_1 - \ell_2) \cdot (\ell_1 + \ell_8 + \ell_6)$	$\ell_1, \ell_2, \ell_5, \ell_6, \ell_8$
2	ℓ_{16}	$(\ell_3 - \ell_0 - \ell_1) \cdot (3\ell_0 + \ell_7)$	$\ell_0, \ell_1, \ell_3, \ell_7$
3	ℓ_{12}	$(\ell_5 - \ell_1 - \ell_2) \cdot (\ell_{10} + \ell_9)$	$\ell_1, \ell_2, \ell_5, \ell_9, \ell_{10}$
3	ℓ_{14}	$(3\ell_0 + \ell_7) \cdot (\ell_{10} + \ell_9)$	$\ell_0, \ell_7, \ell_9, \ell_{10}$

Table: Dependencies of multiplications inside the complete addition formulas

Hardware implementation with 3 multipliers

- platform: Microsemi IGLOO FPGA (low-power)
- area-optimized FPGA architecture of Montgomery multiplication algorithm
- using the pipelined Math blocks and the embedded memory blocks

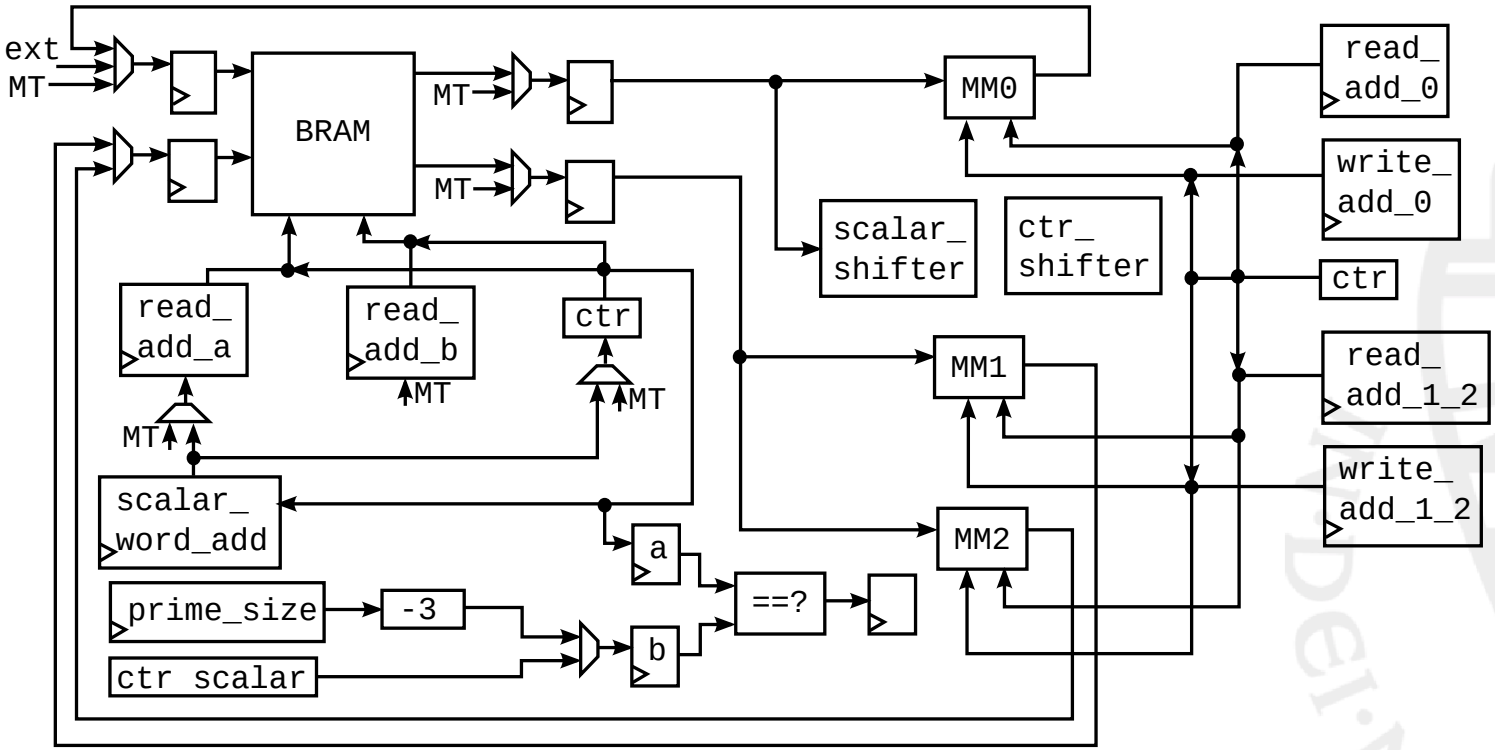
Math Blocks (behave like DSPs)

- can do multiplications in combination with additions and accumulations
- have a total of 3 inputs available to the user, two operands for multiplication and one operand for addition
- possible to negate the multiplication result, shift the result to the rights etc.
- possible to connect the input of one Math Block with the output of a previous one

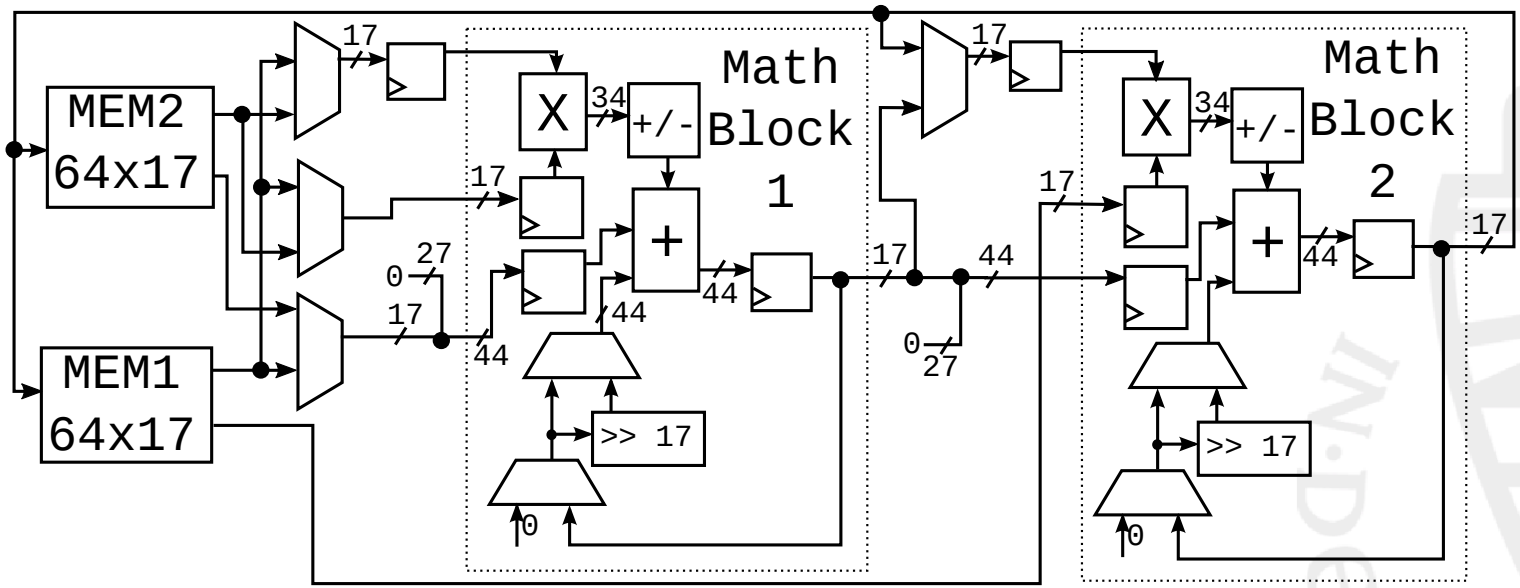
Hardware architecture: overview

- Scalar multiplication built on top of Montgomery processor(s)
- Architecture consists of 3 Montgomery multipliers and a single BRAM block
- Circuit behaves like a three-stage pipeline: operands are fed into the circuit, computed and stored Montgomery processor
  - 2 internal multipliers and 2 memory blocks
  - performing Montgomery multiplication, addition/subtraction without reduction

Complete arhitecture



Montgomery processor



Scheduling for 3 processors

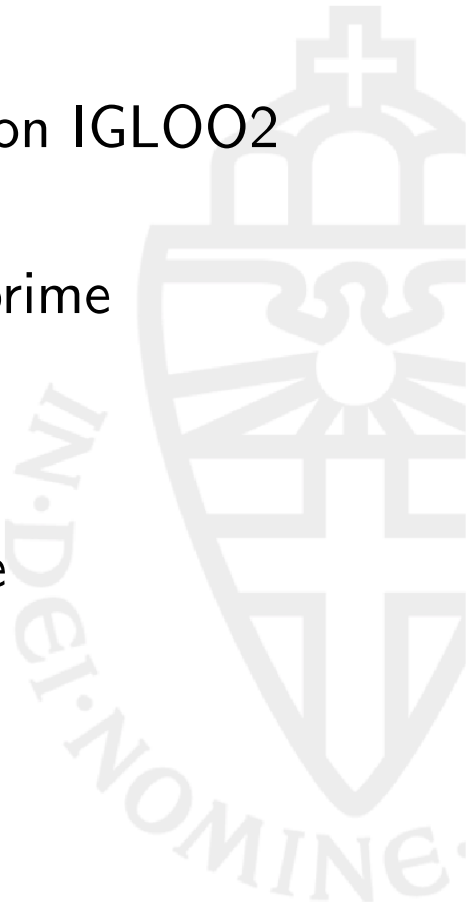
Line in Alg. 2	MM0	MM1	MM2
1	$t_0 \leftarrow X_1 \cdot X_2$	$t_1 \leftarrow Y_1 \cdot Y_2$	
			$t_2 \leftarrow Z_1 \cdot Z_2$
2	$t_3 \leftarrow X_1 + Y_1$	$t_4 \leftarrow X_2 + Y_2$	
			$t_5 \leftarrow Y_1 + Z_1$
3	$t_7 \leftarrow X_1 + Z_1$	$t_8 \leftarrow X_2 + Z_2$	
			$t_6 \leftarrow Y_2 + Z_2$
4	$t_9 \leftarrow t_3 \cdot t_4$	$t_{11} \leftarrow t_7 \cdot t_8$	
			$t_{10} \leftarrow t_5 \cdot t_6$
5	$t_4 \leftarrow t_1 + t_2$	$t_5 \leftarrow t_0 + t_2$	
			$t_3 \leftarrow t_0 + t_1$
6,7,8	$t_6 \leftarrow b_3 \cdot t_2$	$t_8 \leftarrow a \cdot t_2$	
			$t_2 \leftarrow t_9 - t_3$
			$t_3 \leftarrow t_{10} - t_4$
			$t_4 \leftarrow t_{11} - t_5$
			$t_9 \leftarrow t_0 + t_0$
			$t_{10} \leftarrow t_9 + t_0$
9	$t_5 \leftarrow b_3 \cdot t_4$	$t_{11} \leftarrow a \cdot t_4$	
			$t_7 \leftarrow t_0 - t_8$
			$t_9 \leftarrow a \cdot t_7$
10	$t_0 \leftarrow t_8 + t_{10}$	$t_4 \leftarrow t_{11} + t_6$	
			$t_7 \leftarrow t_5 + t_9$
11	$t_5 \leftarrow t_1 - t_4$	$t_6 \leftarrow t_1 + t_4$	
12	$t_4 \leftarrow t_0 \cdot t_7$	$t_1 \leftarrow t_5 \cdot t_6$	
			$t_8 \leftarrow t_3 \cdot t_7$
13	$t_{11} \leftarrow t_0 \cdot t_2$	$t_9 \leftarrow t_2 \cdot t_5$	
			$t_{10} \leftarrow t_3 \cdot t_6$
14	$Y_1 \leftarrow t_1 + t_4$	$X_1 \leftarrow t_9 - t_8$	
			$Z_1 \leftarrow t_{10} + t_{11}$

Comparison

Work	Field	FPGA	Slice/ ALM	LUT	FF	Emb. Mult.	BRAM 64×18	BRAM 1k×18	Freq. (MHz)	Scalar Mult. Cycles	Scalar Mult. (ms)
For all prime fields and prime order short Weierstrass curves											
Our	256	IGLOO 2[4]	–	2828	1048	6	6	1	100	1421312	14.21
For NIST curves [29] only											
[35]	256	SmartFusion[4]	–	3690	3690	0	0	12	109	2103941	19.3
[35]	256	Virtex II Pro[4]	773	1546⋆	1546⋆	1	0	3	210	2103941	10.02
[35]	256	Virtex II Pro[4]	1158	2316⋆	2316⋆	4	0	3	210	949951	4.52
[30]	256	Virtex 5[6]♣	1914	7656⋆	7656⋆	4	0	12	210	830000	3.95
[16]	192	Virtex II Pro[4]	3173	6346⋆	6346⋆	16	0	6	93	920700†	9.90
[32]	256	Spartan 6[6]	72	193	35	8	0	24	156.25	1906250†	12.2
[24]	256	Virtex 4[4]	7020	12435	3545	8	0	4	182	993174†	5.457
[1]	256	Virtex 6[6]♣	11.2k	32.9k	89.6k⋆	289	0	256	100	39922	0.40
[17]	256	Virtex 4[4]	1715	2589	2028	32	0	11	490	303450	0.619
For only Edwards or Twisted Edwards curves											
[2]	192	Spartan 3E [4]	4654	9308⋆	9308⋆	0	0	0	10	125430†	12.543
[34]	256	Zynq[6]♣	1029	2783	3592	20	0	4	200	64770	0.324
For only specific field size, but works with any prime											
[36]	256	Virtex II Pro[4]	1832	3664⋆	3664⋆	2	0	9	108.2	3227993	29.83
[36]	256	Virtex II Pro[4]	2085	4170⋆	4170⋆	7	0	9	68.17	1074625	15.76
[18]	256	Stratix II[4]	9177	18354⋆	18354⋆	96	0	0	157.2	106896†	0.68
[27]	256	Virtex II Pro[4]	15755	31510⋆	31510⋆	256	0	0	39.46	151360	3.86
[25]	256	Virtex 4[4]	4655	5740	4876	37	0	11	250	109297	0.44

Performance results

- hardware architecture supporting 116 to 522 bits on IGLOO2 FPGA
- scalar multiplication takes 8.61 *ms* for a 256-bit prime
- optimizing on Math Blocks
- competitive with other related works
- side-channel resistance should be easier to achieve



Side-channel issues

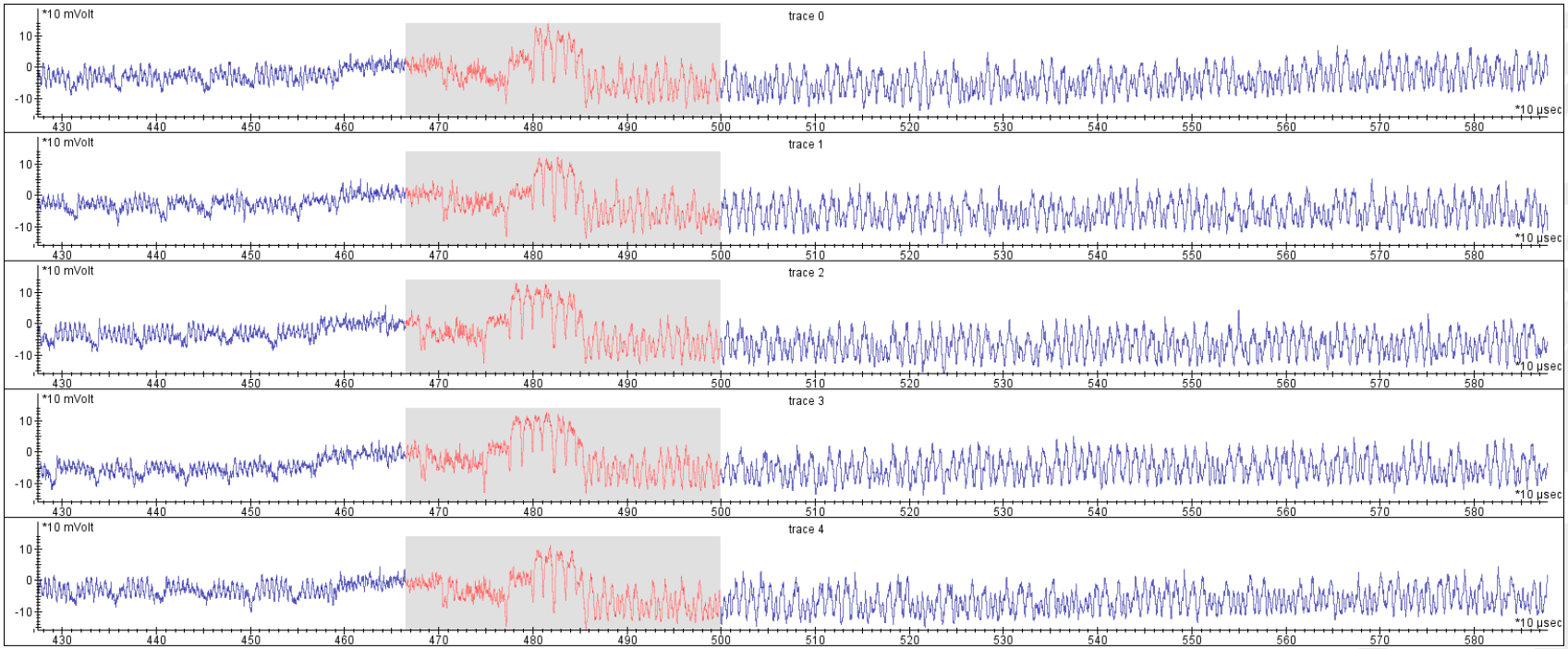
- completeness helps with SPA-resistance
- regular scalar multiplication algorithm
- other countermeasures required for DPA-resistance
  - projective coordinate randomization
  - randomized scalar splitting

Side-channel issues

- completeness helps with SPA-resistance
- regular scalar multiplication algorithm
- other countermeasures required for DPA-resistance
  - projective coordinate randomization
  - randomized scalar splitting

SPA on ECC scalar multiplication

5 traces of the first round of Lim-Lee algorithm. Pattern: 11001

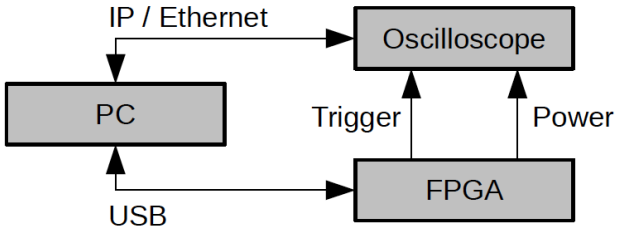


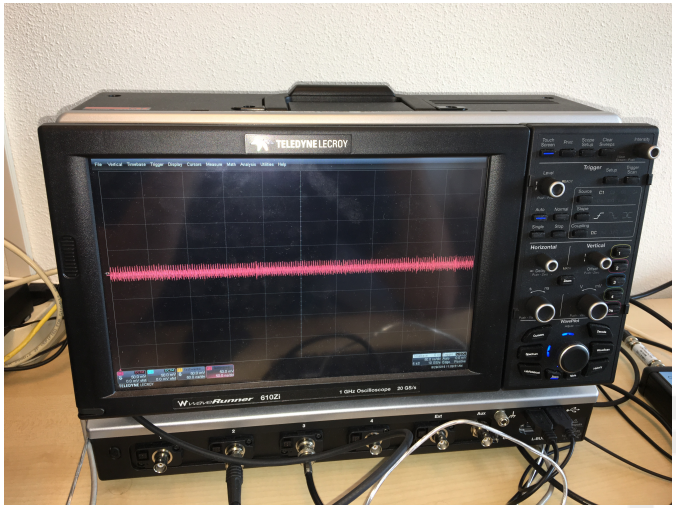
Slide credit: L. Chmielewski.

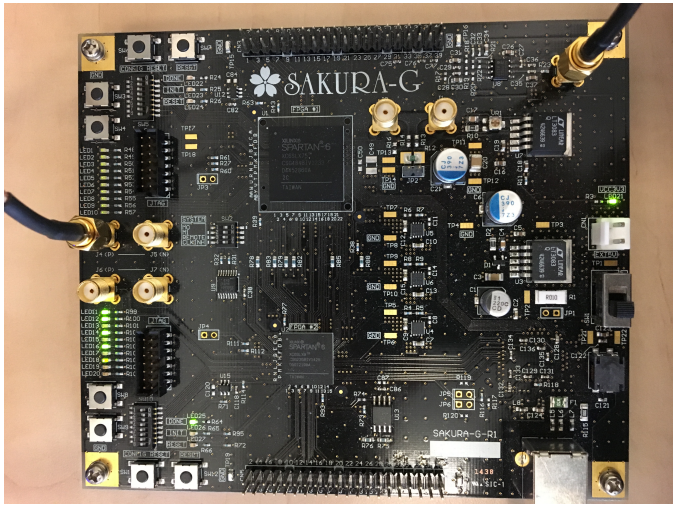
Side-channel analysis of complete formula

- considered 3 implementations: insecure + 2 countermeasures (randomized Z and random scalar splitting)
- Montgomery ladder used for scalar multiplication
- slightly different hardware architecture used for analysis

Setup

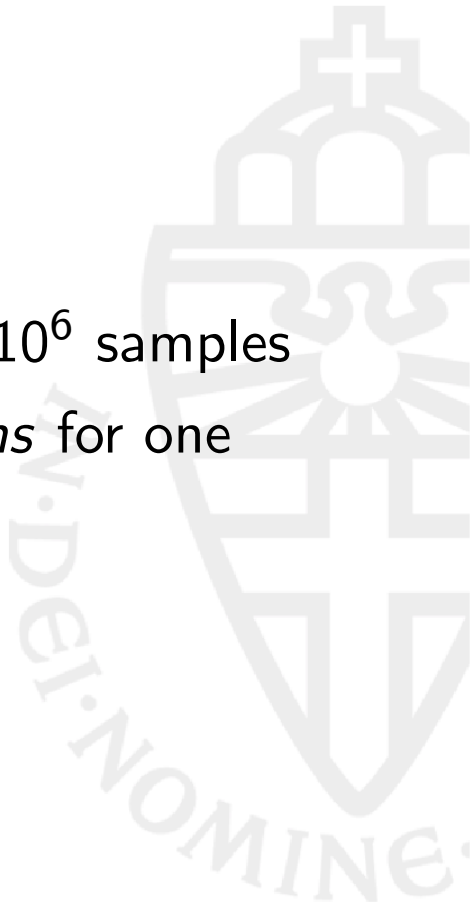

IP / Ethernet
Oscilloscope
PC
Trigger
Power
FPGA
USB


TELEDYNE LECROY


SAKURA-G
SAKURA-G-R1

Setup details

- scope used is LecroyWaverunner 610Zi
- SAKURA-GII board
- configured with 10^8 samples/s and at most 32×10^6 samples
- crypto core running at 6 MHz, takes about 300 *ms* for one point multiplication



Test Vector Leakage Assessment (TVLA) analysis

- with no countermeasures applied
  strong leakage
- with coordinate randomization
  less evidence leakage but could be exploitable
- with coordinate randomization and random scalar splitting
  no evidence of leakage

Contributions

- The first hardware implementation of the complete formulas for Weierstrass curves
- A detailed FPGA implementation with three arithmetic units
- Algorithms for architectures from 2 to 6 processors
- Side-channel evaluation of the complete formula
  - on a compact hardware architecture
  - for NIST standardized P-256 curve
  - two countermeasures evaluated

Thanks

Thanks for your attention!