Hardware Architectures Exploration for Hyper-Elliptic Curve Cryptography

Gabriel GALLIN and Arnaud TISSERAND

CNRS – IRISA – Lab-STICC
HAH Project

Crypto'Puces May, 29th - June, 2nd 2017

Summary

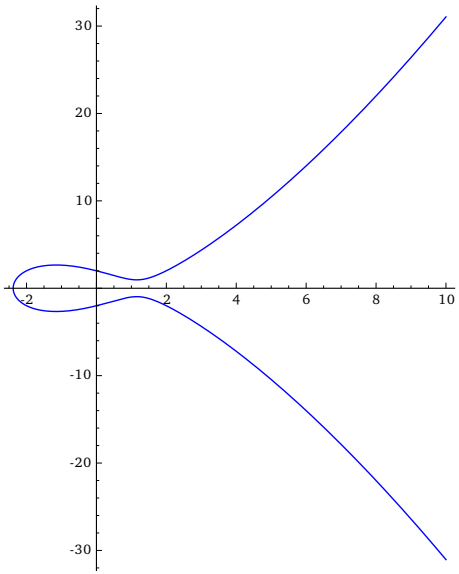
Summary

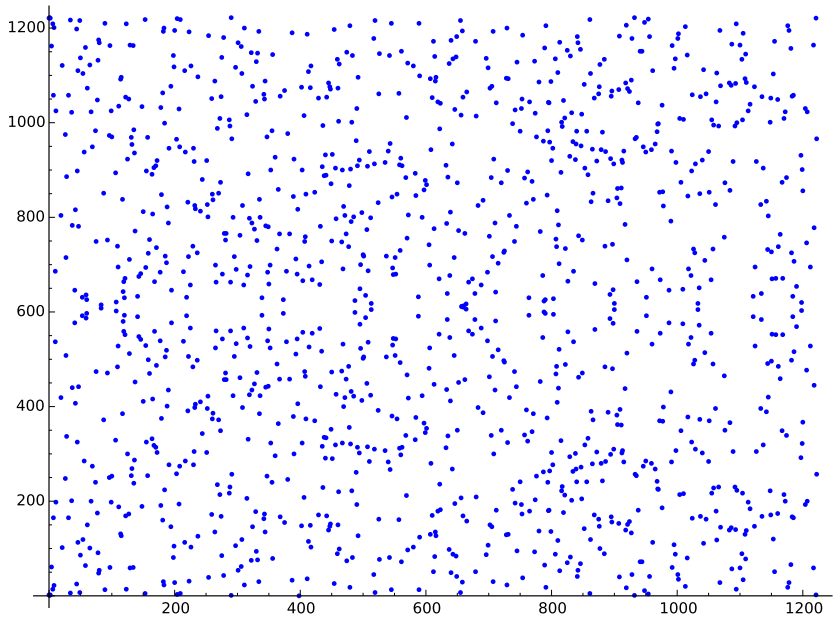
Public-Key Cryptography (PKC)

- Cryptographic primitives for protocols such as digital signature, key exchange and some specific encryption schemes
- First PKC standard: RSA
  - Large keys ($\geq$ 2000 bits recommended today)
  - Too costly for embedded applications
- Elliptic Curve Cryptography (ECC):
  - Actual standard for public key crypto-systems
  - Better performance and lower cost than RSA
- Hyper-Elliptic Curve Cryptography (HECC):
  - Evolution of ECC focusing on larger set of curves
  - Studied for future generations of asymmetric crypto-systems

Elliptic and Hyper-Elliptic Curves

- Elliptic Curves
  - Equation (Weierstrass) $E/\mathbb{K} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$
  - Defined over field $\mathbb{K}$: real numbers ($\mathbb{R}$), prime finite field ($\mathbb{F}_\mathcal{P}$ or $GF(p)$)
  - Coefficients and coordinates size in $\mathbb{F}_\mathcal{P}$: $200 \sim 300$ bits





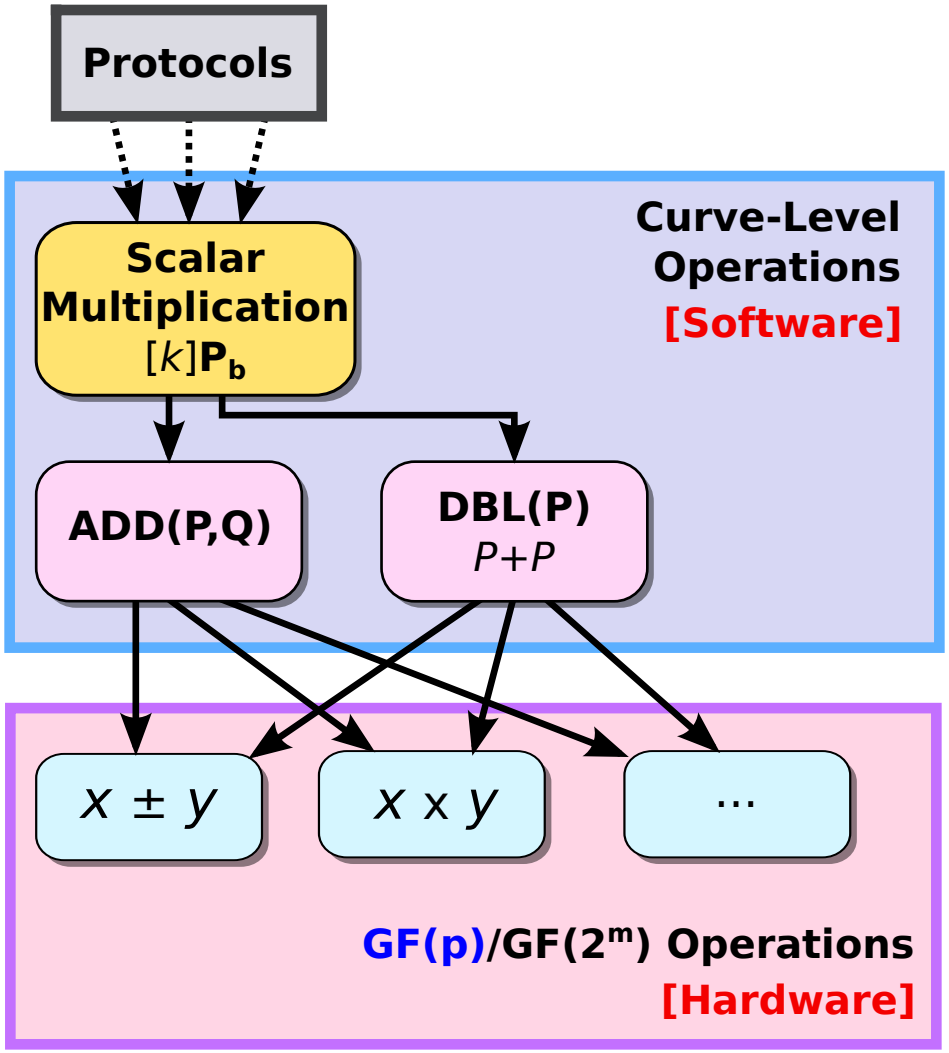
Elliptic and Hyper-Elliptic Curves

- Elliptic Curves
  - Equation (Weierstrass) $E/\mathbb{K} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$
  - Defined over field $\mathbb{K}$: real numbers ($\mathbb{R}$), prime finite field ($\mathbb{F}_\mathcal{P}$ or $GF(p)$)
  - Coefficients and coordinates size in $\mathbb{F}_\mathcal{P}$: $200 \sim 300$ bits
- Hyper-Elliptic Curves
  - **Much more complex!!!**
  - Equation $H/\mathbb{K} : y^2 + h(x)y = f(x)$, $deg(h) < g$ and $deg(f) = 2g + 1$
  - g: genus of the curve, $g \leq 2$ in practice for reliable HECC
  - Coefficients and coordinates size in $\mathbb{F}_\mathcal{P}$: $100 \sim 200$ bits

Elliptic and Hyper-Elliptic Curves

- Elliptic Curves
  - Equation (Weierstrass) $E/\mathbb{K} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$
  - Defined over field $\mathbb{K}$: real numbers ($\mathbb{R}$), prime finite field ($\mathbb{F}_\mathcal{P}$ or $GF(p)$)
  - Coefficients and coordinates size in $\mathbb{F}_\mathcal{P}$: $200 \sim 300$ bits
- Hyper-Elliptic Curves
  - **Much more complex!!!**
  - Equation $H/\mathbb{K} : y^2 + h(x)y = f(x)$, $deg(h) < g$ and $deg(f) = 2g + 1$
  - g: genus of the curve, $g \leq 2$ in practice for reliable HECC
  - Coefficients and coordinates size in $\mathbb{F}_\mathcal{P}$: $100 \sim 200$ bits
- Kummer surface
  - Not an additive group: no addition law
  - Can be used in HECC using some *(magic)* trick
  - Reduced complexity for curve operations

Operations Hierarchy in (H)ECC



Metric for algorithms efficiency: number of multiplications ($\mathtt{M}$) and squares ($\mathtt{S}$) in $\mathbb{F}_{\mathcal{P}}$

ECC versus HECC

	size of $\mathbb{F}_{\mathcal{P}}$	ADD	DBL	source
ECC	ℓ_{ECC}	$12\mathtt{M} + 2\mathtt{S}$	$7\mathtt{M} + 3\mathtt{S}$	[EFD]
HECC	$\ell_{\mathrm{HECC}} \approx \frac{1}{2}\ell_{\mathrm{ECC}}$	$40\mathtt{M} + 4\mathtt{S}$	$38\mathtt{M} + 6\mathtt{S}$	[Lange, 2005]
Kummer	ℓ_{HECC}	$19\mathtt{M} + 12\mathtt{S}$		[Renes et al., 2016]

- ECC:
  - Size of $\mathbb{F}_{\mathcal{P}}$ elements $2\times$ larger
  - Simpler ADD and DBL operations
- HECC:
  - Smaller $\mathbb{F}_{\mathcal{P}}$
  - More operations in $\mathbb{F}_{\mathcal{P}}$ for ADD and DBL
- [Renes et al., 2016]: HECC using Kummer is more efficient than ECC
  - Software implementation on small processors
  - ARM Cortex M0: up to 75% clock cycles reduction for signatures
  - AVR AT-mega: up to 32% cycles reduction for Diffie-Hellman

Arithmetic in $\mathbb{F}_{\mathcal{P}}$

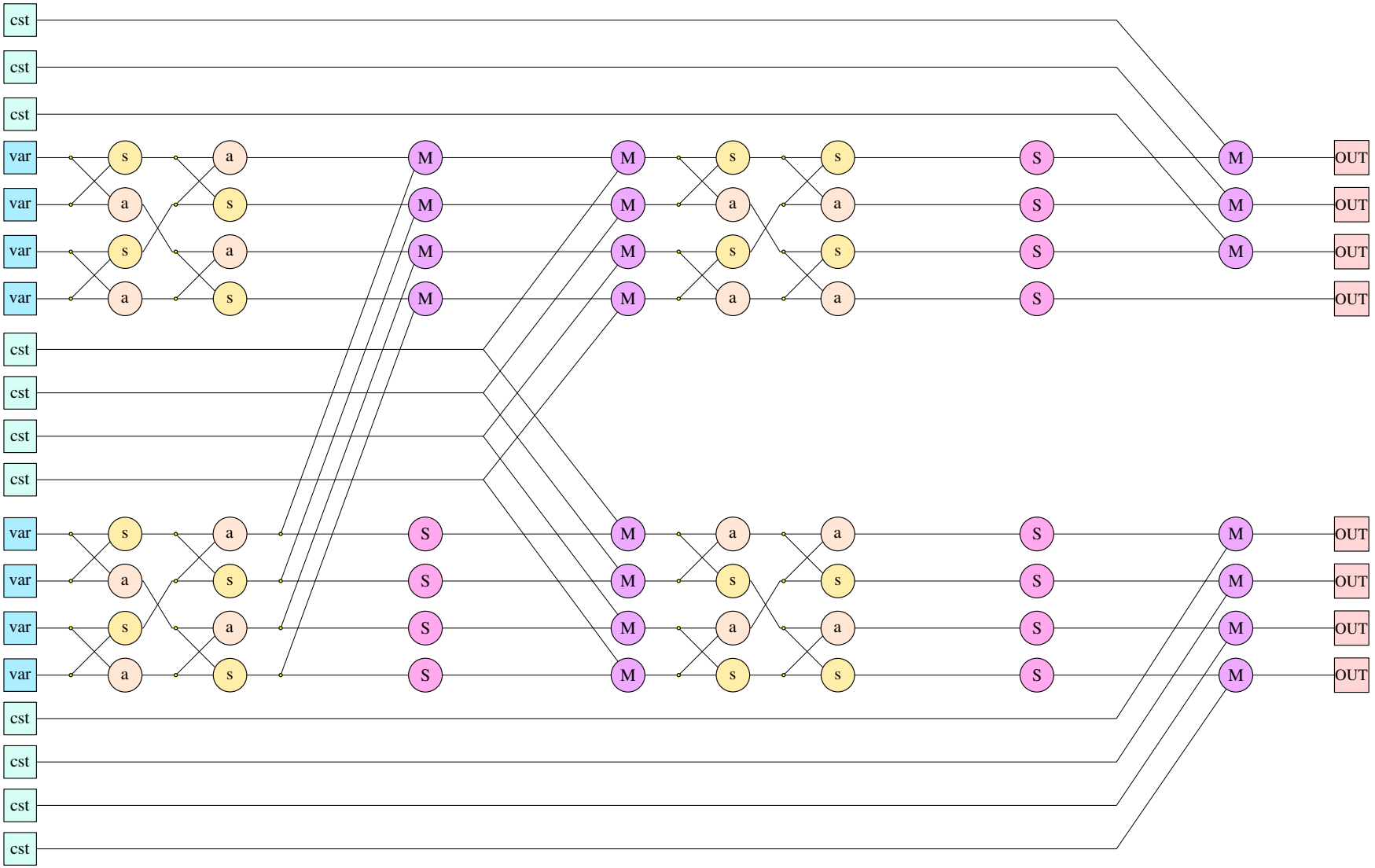
- Curve operations (`ADD`, `DBL`) are built with finite field operations
- Modular arithmetic in $\mathbb{F}_{\mathcal{P}}$:
  - Size of elements: $100 \sim 200$ bits for HECC
  - Operations involve modular reduction
  - Choice of $\mathcal{P}$:
    - Generic $\mathcal{P}$: more flexible but slower
    - Specific $\mathcal{P}$ (*e.g.* pseudo-Mersenne): faster but more specific
- Modular multiplication and square:
  - Most common and costly operations
  - Dedicated arithmetic unit(s) must be efficient

Summary

Curve-Level Operations in Kummer

- $\mathcal{K}_\mathcal{C}$: projection in Kummer of hyper-elliptic curve $\mathcal{C}$
- No `ADD` operation in $\mathcal{K}_\mathcal{C}$, but still `DBL`
- *Differential addition* operation:
  $\texttt{xADD}(\pm P, \pm Q, \pm(P-Q)) \rightarrow \pm(P+Q)$
- `xADD` and `DBL` can be combined:
  $\texttt{xDBLADD}(\pm P, \pm Q, \pm(P-Q)) \rightarrow (\pm[2]P, \pm(P+Q))$
- Operations presented in [Renes et al., 2016]
  See also [Gaudry, 2007] and [Bos et al., 2016] for more details

xDBLADD $\mathbb{F}_{\mathcal{P}}$ Operations



Scalar Multiplication

Montgomery ladder based *crypto_scalarmult* [Renes et al., 2016]

Require: m-bit scalar $k = \sum_{i=0}^{m-1} 2^i k_i$, point P_b, *constant* $\in \mathbb{F}_{\mathcal{P}}^4$
Ensure: $V_1 = [k]P_b$, $V_2 = [k+1]P_b$
  $V_1 \leftarrow$ *constant*
  $V_2 \leftarrow P_b$
  for $i = m-1$ **downto** 0 **do**
    $(V_1, V_2) \leftarrow \texttt{CSWAP}(k_i, (V_1, V_2))$
    $(V_1, V_2) \leftarrow \texttt{xDBLADD}(V_1, V_2, P_b)$
    $(V_1, V_2) \leftarrow \texttt{CSWAP}(k_i, (V_1, V_2))$
  end for
  return (V_1, V_2)

$\texttt{CSWAP}(k_i, (X, Y))$ returns (X, Y) if $k_i = 0$, else (Y, X)

Key Ideas

- Scalar multiplication in Kummer HECC based on Montgomery ladder
  - Constant time
  - Uniform operations $\rightarrow$ independent from scalar (key) bits

- Scalar multiplication core operations: `xDBLADD` and `CSWAP`
  - `xDBLADD`: $19\mathtt{M} + 12\mathtt{S}$
  - Some parallelism between `xDBLADD` internal $\mathbb{F}_{\mathcal{P}}$ operations
  - `CSWAP`: very simple but involves secret bits (to be protected)

Summary

Motivations

- We target FPGA implementations
- Kummer HECC operations $\rightarrow$ internal parallelism:
  - Large number of solutions for HECC architectures
  - Large exploration space: number/types of blocks, architecture models, types of internal communications/memory, control, ...
- Developping a complete accelerator in HDL is long and complex
  - Impossible to implement all solutions in the design space
  - Exploration tools required to quickly evaluate various architectures and parameters
  - Only the most interesting architectures will be implemented
- Hardware implementations
  - Small & low-cost FPGAs (Xilinx Spartan 6)
  - Efficiency: computation time, area cost, energy, side-channel leakage

Architectures Exploration Concerns

- Constraints:
  - Complex and large operations
  - Need for numerical validation (and debug)
  - Various architectures types and parameters
    - Type of arithmetic algorithms
    - Number of blocks
    - Type of internal communications and control
- Solution:
  - Hierarchical description and simulation at CCABA level (Critical-Cycle Accurate, Bit Accurate)
  - Blocks: developped in HDL (perfectly known behaviour)
  - Architecture: high-level description
    - Many models
    - Many parameters

A Simulator to Explore Architectures

- We developped a simulator dedicated to HECC architectures
- Description and simulation at CCABA level
  - Control signals and communications between blocks must be accurately modeled
  - A scalar multiplication takes several thousand cycles
  - Only critical cycles in blocks (inputs, outputs and control) are used in the architecture
- Software simulator coded in Python
  - Fast developpement
  - Provides interface with *Sagemath*[1] tool for numerical validation and HECC support

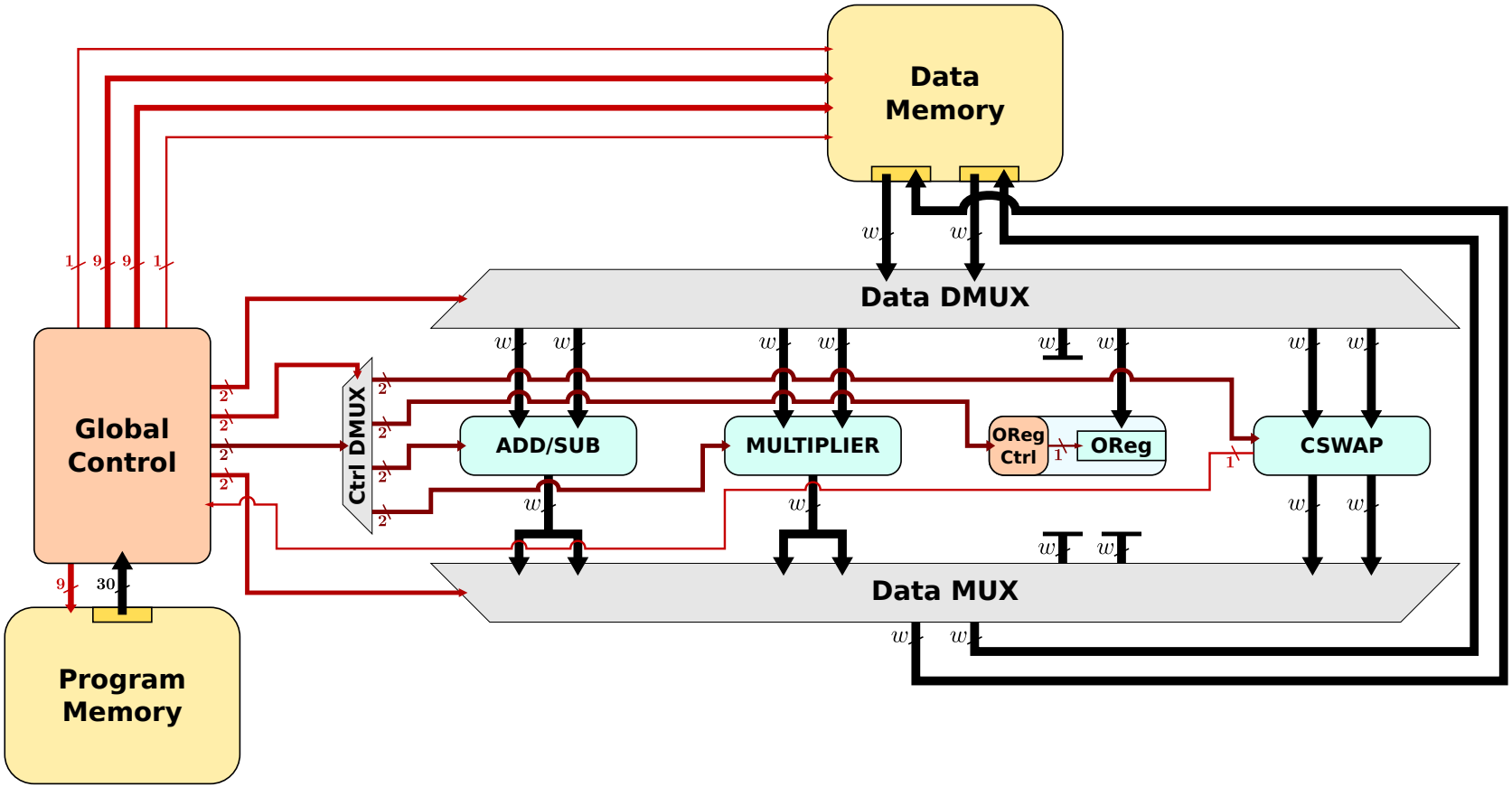
[1] cf. http://www.sagemath.org/

Modelisation of Architectures

- Architecture models built over a set of blocks
  - Arithmetic units: adders, subtractors, multipliers, ...
  - Memories, registers, ...
  - Interconnect: buses, multiplexers, ...
- Blocks models:
  - All inputs/outputs are bit accurate
  - All inputs/outputs and control signals are cycle accurate
  - Blocks characteristics come from FPGA implementation (latency, area cost, ...)
- Control from curve operations formulas and units configurations
  - Manage activity of blocks in the architecture
  - Manage all internal communications
  - Developped tool for scheduling $\mathbb{F}_{\mathcal{P}}$ operations (*work in progress*)

Summary

A First Architecture Model



Parameters specified at design time:

- Width w for internal communications ($s \times w = n$)
- Types and number of blocks

Functional Units Configuration

- Modular adder-subtractor unit (ADDSUB):
  - One unit for two types of operation
  - Pipelined operator (4 cycles latency)
  - Delay: $8 \sim 11$ cycles depending on external datapath width
- Modular multiplier unit (HTMM):
  - Hyper-threaded multiplier: 3 sets of operands computed in parallel
  - Delay: $68 \sim 71$ cycles depending on external datapath width
- CSWAP unit:
  - Manage parsing of key bits
  - Delay: $2 \sim 4$ cycles depending on external datapath width

Results for the Basic Architecture

Implem. ($s \times w$)*	number of cycles	Functional Unit	DSP	BRAM	FF	LUT	Slices	RAM size (num. lines)
4x34	207383	HTMM	11	2	587	359	180	12
		ADDSUB	0	0	366	226	80	-
		DATA_MEM	0	1	0	0	0	112
		PRGM_MEM	0	1	0	0	0	208
		CSWAP	0	0	536	290	103	-
2x68	185615	HTMM	11	2	970	633	315	12
		ADDSUB	0	0	713	382	148	-
		DATA_MEM	0	2	0	0	0	56
		PRGM_MEM	0	1	0	0	0	234
		CSWAP	0	0	553	297	122	-
1x136	183051	HTMM	11	2	1066	623	309	12
		ADDSUB	0	0	784	464	212	-
		DATA_MEM	0	4	0	0	0	26
		PRGM_MEM	0	1	0	0	0	250
		CSWAP	0	0	685	431	155	-

* s: number of words, w: size of words

Increasing the Number of Units

Implem. ($s \times w$)*	number of cycles	Functional Unit	DSP	BRAM	FF	LUT	Slices	RAM size (num. lines)
4x34	**203543**	HTMM **x 2**	**22**	**4**	**1174**	**718**	**360**	12
		ADDSUB **x 2**	0	0	**732**	**452**	**160**	-
		DATA_MEM	0	1	0	0	0	**108**
		PRGM_MEM	0	1	0	0	0	**213**
		CSWAP	0	0	536	290	103	-
2x68	**125455**	HTMM **x 2**	**22**	**4**	**1940**	**1266**	**630**	12
		ADDSUB **x 2**	0	0	**1426**	**764**	**296**	-
		DATA_MEM	0	4	0	0	0	**50**
		PRGM_MEM	0	1	0	0	0	**211**
		CSWAP	0	0	553	297	122	-
1x136	**115211**	HTMM **x 2**	**22**	**4**	**2132**	**1246**	**618**	12
		ADDSUB **x 2**	0	0	**1568**	**928**	**424**	-
		DATA_MEM	0	4	0	0	0	**25**
		PRGM_MEM	0	1	0	0	0	**235**
		CSWAP	0	0	685	431	155	-

* s: number of words, w: size of words

Conclusions and Perspectives

- Kummer based HECC is an efficient alternative to ECC
  - More complex formulas, with some internal parallelism
  - Large space of solutions for HECC architectures
- We designed a CCABA simulator for architectures exploration
  - Hierarchical description of each architecture
    - Blocks described in HDL, only critical cycles are modeled
    - High-level representation of the architecture
  - Fast (numerical) validation and evaluation of solutions
- Perspectives and future work
  - Improving the simulator
    - Study new algorithms for scheduling $\mathbb{F}_{\mathcal{P}}$ operations
    - Work on automated generation of HDL code from high-level description
  - Explore new architectural solutions
    - Implement most interesting solutions
    - Evaluate the impact of each solution on power signature

This work is funded by **HAH** project http://h-a-h.inria.fr/

Thank you for your attention

References I

[EFD] Bernstein, D. J. and Lange, T.
Explicit-formulas database.
http://hyperelliptic.org/EFD/.

[Bos et al., 2016] Bos, J. W., Costello, C., Hisil, H., and Lauter, K. (2016).
Fast cryptography in genus 2.
Journal of Cryptology, 29(1):28–60.

[Cohen et al., 2005] Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., and Vercauteren, F. (2005).
Handbook of Elliptic and Hyperelliptic Curve Cryptography.
Discrete Mathematics and Its Applications. Chapman & Hall/CRC.

[Gaudry, 2007] Gaudry, P. (2007).
Fast genus 2 arithmetic based on theta functions.
Journal of Mathematical Cryptology, 1(3):243–265.

[Hankerson et al., 2004] Hankerson, D., Menezes, A., and Vanstone, S. (2004).
Guide to Elliptic Curve Cryptography.
Springer.

[Lange, 2005] Lange, T. (2005).
Formulae for Arithmetic on Genus 2 Hyperelliptic Curves.
Applicable Algebra in Engineering, Communication and Computing, 15(5):295–328.

[Ma et al., 2013] Ma, Y., Liu, Z., Pan, W., and Jing, J. (2013).
A high-speed elliptic curve cryptographic processor for generic curves over GF(p).
In *Proc. 20th International Workshop on Selected Areas in Cryptography (SAC)*, volume 8282 of *LNCS*, pages 421–437.
Springer.

References II

[Montgomery, 1987] Montgomery, P. L. (1987).
Speeding the Pollard and elliptic curve methods of factorization.
Mathematics of Computation, 48(177):243–264.

[Renes et al., 2016] Renes, J., Schwabe, P., Smith, B., and Batina, L. (2016).
μKummer: Efficient hyperelliptic signatures and key exchange on microcontrollers.
In *Proc. Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 9813 of *LNCS*, pages 301–320. Springer.

ADD and DBL in ECC – an Illustration

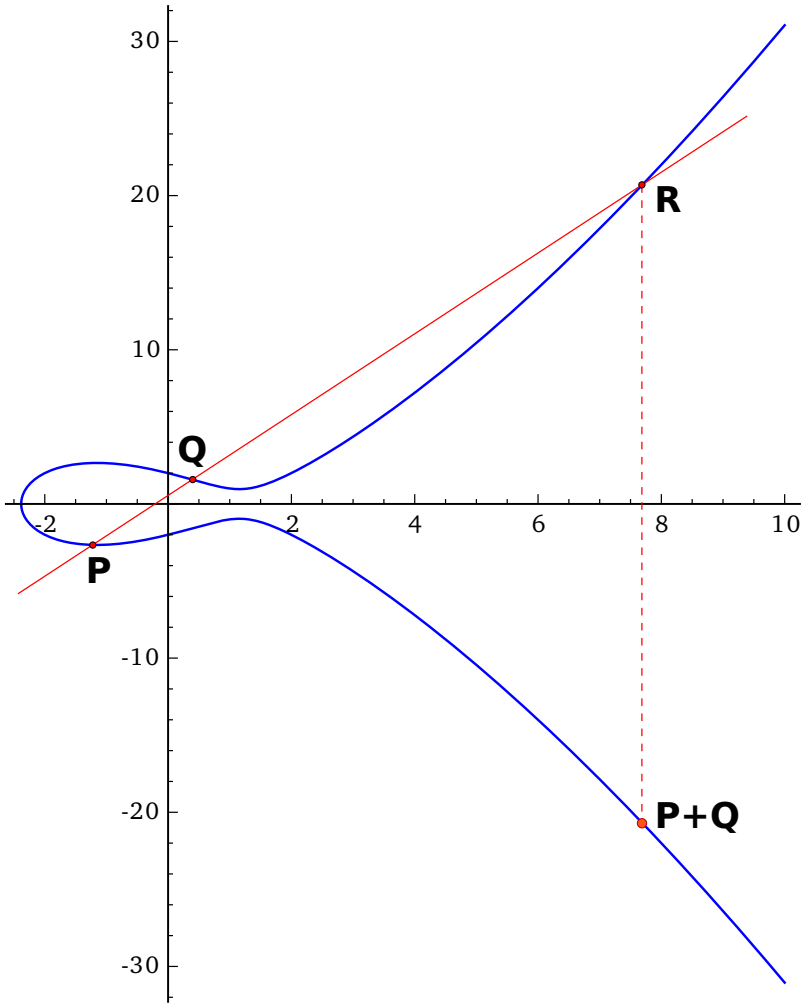


Figure : ADD(**P**, **Q**)

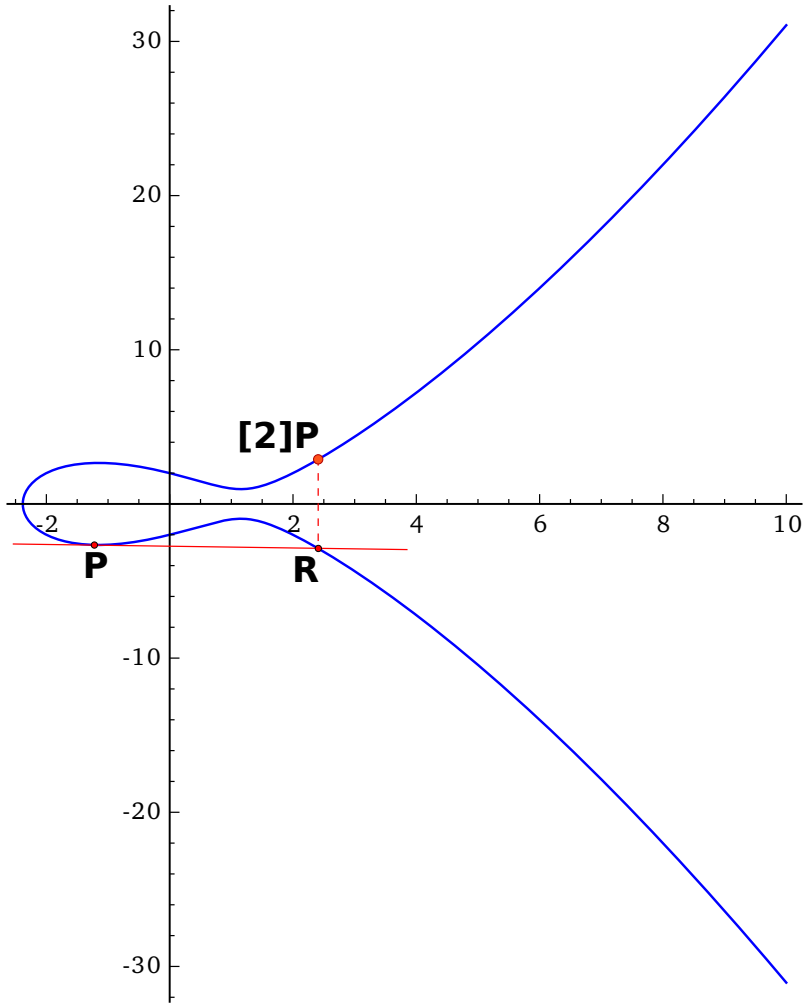


Figure : DBL(**P**)

HTMM Internal Architecture

- Based on Montgomery modular multiplication algorithm
  - Iterations over partial products and reductions
  - 3 internal partial products to compute in each iteration
- HTMM architecture: 3 hardware stages
  - One stage for each internal partial product
  - Stages are fully pipelined (several clock cycles per stage)
  - 3 to 4 DSP slices in each stage