Software implementation of binary elliptic curves: impact of the carry-less multiplier on scalar multiplication

J. Taverne[0], A. Faz-Hernández[1], D. F. Aranha[2],
F. Rodríguez-Henríquez[1], D. Hankerson[3], J. López[2]

[0]Université Lyon 1 - ISFA - France
[1]CINVESTAV - IPN - México
[2]Universidad de Campinas - Brazil
[3]Auburn University - USA

CHES - Nara - Japan
September 29th 2011

Outline of the talk

1 Introduction

2 Algorithms and implementation
- Binary field arithmetic
- Elliptic curves arithmetic
- Scalar multiplication

3 Results

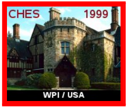
Outline of the talk

1 Introduction

2 Algorithms and implementation
- Binary field arithmetic
- Elliptic curves arithmetic
- Scalar multiplication

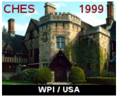
3 Results

Scalar multiplication implementation at CHES



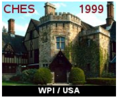
- Julio López, Ricardo Dahab: Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation
- A whole section devoted to Implementation of Elliptic Curve Cryptosystems
- Two sections on ECC : Elliptic Curve Algorithms and Side Channel Attacks on Elliptic Curve Cryptanalysis
- Three sections on Elliptic Curve Cryptography
- ...
- Three papers on efficient/fast ECC implementation

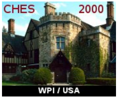
Scalar multiplication implementation at CHES



- Julio López, Ricardo Dahab: Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation
- A whole section devoted to Implementation of Elliptic Curve Cryptosystems
- Two sections on ECC : Elliptic Curve Algorithms and Side Channel Attacks on Elliptic Curve Cryptanalysis
- Three sections on Elliptic Curve Cryptography
- ...
- Three papers on efficient/fast ECC implementation

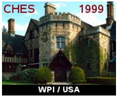
Scalar multiplication implementation at CHES

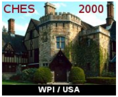




- Julio López, Ricardo Dahab: Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation
- A whole section devoted to Implementation of Elliptic Curve Cryptosystems
- Two sections on ECC : Elliptic Curve Algorithms and Side Channel Attacks on Elliptic Curve Cryptanalysis
- Three sections on Elliptic Curve Cryptography
- ...
- Three papers on efficient/fast ECC implementation

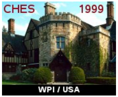
Scalar multiplication implementation at CHES

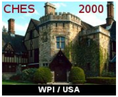




- Julio López, Ricardo Dahab: Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation
- A whole section devoted to Implementation of Elliptic Curve Cryptosystems
- Two sections on ECC : Elliptic Curve Algorithms and Side Channel Attacks on Elliptic Curve Cryptanalysis
- Three sections on Elliptic Curve Cryptography
- ...
- Three papers on efficient/fast ECC implementation

Scalar multiplication implementation at CHES

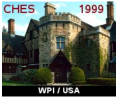


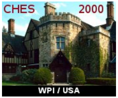


...

- Julio López, Ricardo Dahab: Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation
- A whole section devoted to Implementation of Elliptic Curve Cryptosystems
- Two sections on ECC : Elliptic Curve Algorithms and Side Channel Attacks on Elliptic Curve Cryptanalysis
- Three sections on Elliptic Curve Cryptography
- ...
- Three papers on efficient/fast ECC implementation

Scalar multiplication implementation at CHES





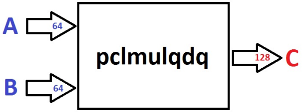
...


- Julio López, Ricardo Dahab: Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation
- A whole section devoted to Implementation of Elliptic Curve Cryptosystems
- Two sections on ECC : Elliptic Curve Algorithms and Side Channel Attacks on Elliptic Curve Cryptanalysis
- Three sections on Elliptic Curve Cryptography
- ...
- Three papers on efficient/fast ECC implementation

Intel® Carry-Less Multiplication Instruction

- Available since Westmere architecture [32nm], `PCLMULQDQ` instruction performs a multiplication of two 64-bit operands without carry bits. Its latency ranges from 10 to 15 clock cycles (8 to 14 in Sandy Bridge).

- Unlike Westmere, new Sandy Bridge architecture provides three-operand code for this instruction.



Impact on the field arithmetic assumptions

Mul in $\mathbb{F}_{2^{233}}$	comb method (LD)	Karatsuba (CMUL)	⇩
Westmere i5	256 cc *	128 cc *	

* according to our experimentations

- Impact on the ratios of multiplication with other operations:
  - Mul/Sq, Mul/Sqrt
  - Inv/Mul
  - Quadratic solver/Mul

- Consequences on the elliptic curves arithmetic:

??? DOUBLING [Multiplication]

HALVING [Quad. solver] ???

Impact on the field arithmetic assumptions

Mul in $\mathbb{F}_{2^{233}}$	comb method (LD)	Karatsuba (CMUL)
Westmere i5	256 cc *	128 cc *

* according to our experimentations

- Impact on the ratios of multiplication with other operations:
  - Mul/Sq, Mul/Sqrt
  - Inv/Mul
  - Quadratic solver/Mul

- Consequences on the elliptic curves arithmetic:

??? DOUBLING [Multiplication]

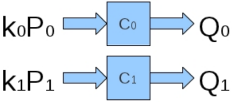
HALVING [Quad. solver] ???

Motivation

- From the point of view of software implementations binary elliptic curves have almost always be considered [much] slower than prime field multiplications.
- Until now, little attention has been put on multi-core implementation of a single elliptic curve scalar multiplication

Motivation

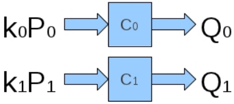
- From the point of view of software implementations binary elliptic curves have almost always be considered [much] slower than prime field multiplications.
- Until now, little attention has been put on multi-core implementation of a single elliptic curve scalar multiplication

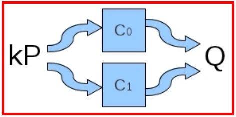


Motivation

- From the point of view of software implementations binary elliptic curves have almost always be considered [much] slower than prime field multiplications.

- Until now, little attention has been put on multi-core implementation of a single elliptic curve scalar multiplication





Our contribution

- Squaring and square-root are not negligible anymore with respect to multiplication
- Half-trace is computed at the same cost as multiplication
- **Fastest single-core implementation** of a single scalar multiplication on various binary curves at the 112- 128- 192-bit security levels
- **Efficient multi-core implementation** of a single scalar multiplication achieving an almost 2 factor of acceleration from algorithm analysis and **1.46 to 1.72** in practice

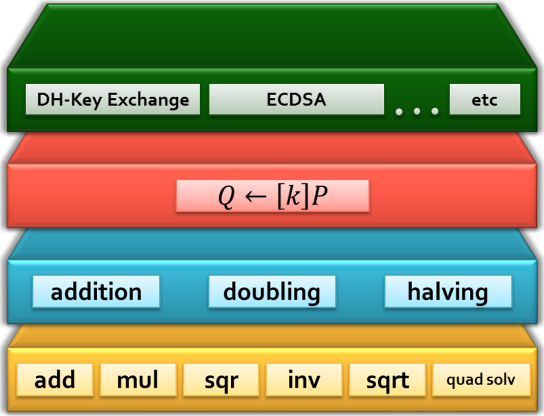
Outline of the talk

1 Introduction

2 Algorithms and implementation
- Binary field arithmetic
- Elliptic curves arithmetic
- Scalar multiplication

3 Results

Structure



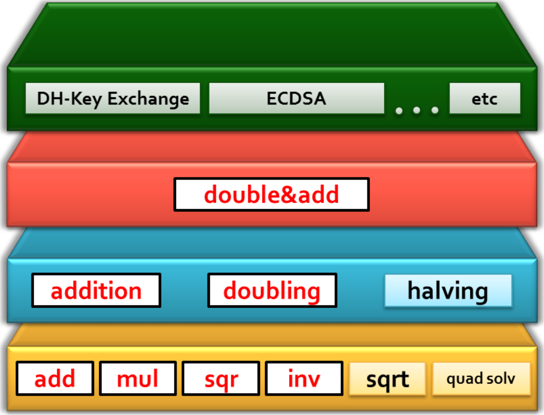
Protocols

Scalar mul.

Ell. curves arith.

$\mathbb{F}_{2^p}$

Structure



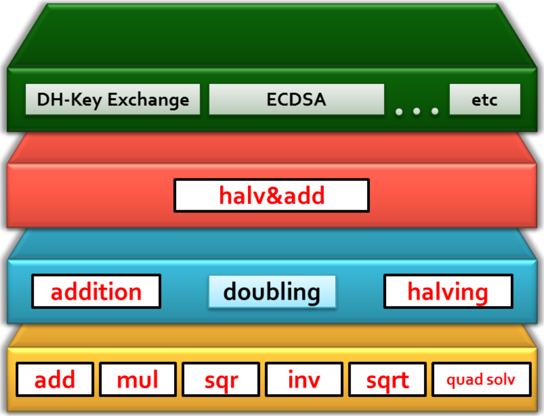
Protocols

Scalar mul.

Ell. curves arith.

$\mathbb{F}_{2^p}$

Structure



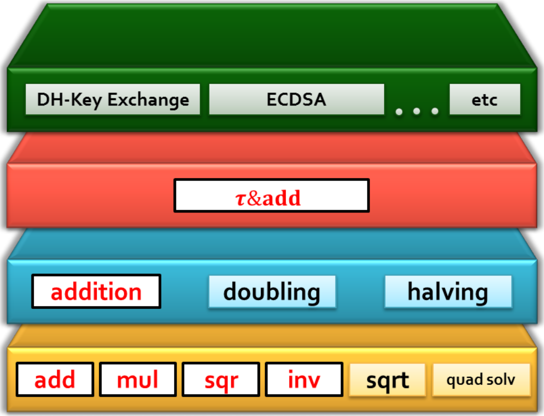
Protocols

Scalar mul.

Ell. curves arith.

$\mathbb{F}_{2^p}$

Structure



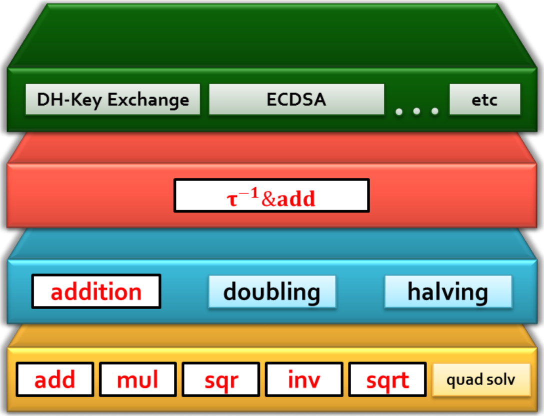
Protocols

Scalar mul.

Ell. curves arith.

$\mathbb{F}_{2^p}$

Structure



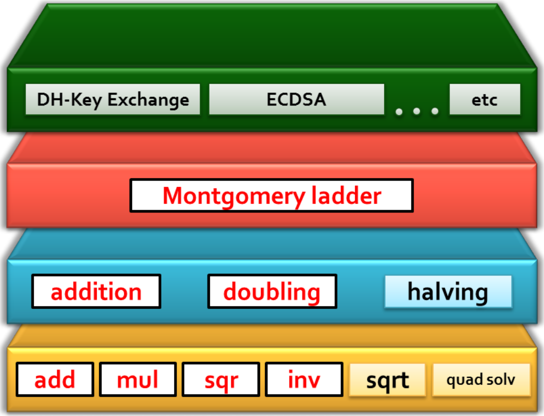
Protocols

Scalar mul.

Ell. curves arith.

$\mathbb{F}_{2^p}$

Structure



Protocols

Scalar mul.

Ell. curves arith.

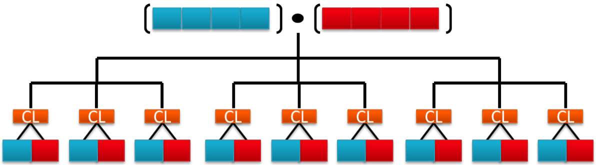
$\mathbb{F}_{2^p}$

Outline of the talk

1. Introduction
2. Algorithms and implementation
   - Binary field arithmetic
   - Elliptic curves arithmetic
   - Scalar multiplication
3. Results

Multiplication

- The maximum amount of work should be done in registers to avoid costly load/store instructions
- The multiplier should have 128-bit granularity to benefit from cheap `xor` and `shift-by-byte` instructions
- Minimal overhead when implementing Karatsuba in $\mathbb{F}_{2^p}$



Squaring and square-root

- Vectorized implementations with simultaneous table lookups through byte shuffling instructions [Aranha et al., LATINCRYPT 10'] improved by a careful reordering of the instructions
- Multi-squaring = exponentiation to 2^k. The method uses only `xor` operations, taking the values from a large precomputed table. Although very memory demanding, this multi-squaring function brings substantial speed improvements.

Quadratic solver $z^2 + z = c$

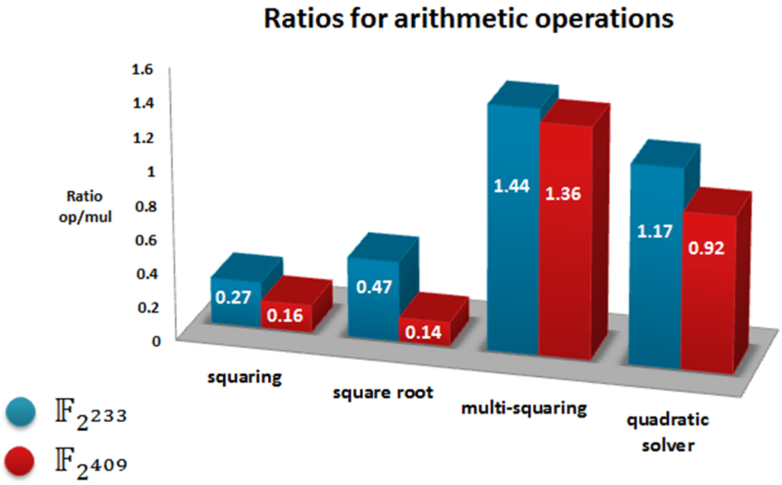
Algorithm 1 Solve $x^2 + x = c$ [Avanzi, IACR ePrint 07']

Input: $c = \sum_{i=0}^{m-1} c_i z^i \in \mathbb{F}_{2^m}$ where m is odd and $\mathrm{Tr}(c) = 0$
Output: a solution s of $x^2 + x = c$

1: compute $H(l_0 c^{8i+1} + l_1 c^{8i+3} + l_2 c^{8i+5} + l_3 c^{8i+7})$ for $i \in I = \{0, \ldots, \lfloor \frac{m-3}{8} \rfloor\}$ and $l_j \in \mathbb{F}_2$
2: $s \leftarrow 0$
3: **for** $i = (m-1)/2$ **downto** 1 **do**
4:     **if** $c_{2i} = 1$ **then**
5:         $c \leftarrow c + z^i$, $s \leftarrow s + z^i$
6: **return** $s \leftarrow s + \sum_{i \in I} c^{8i+1} H(z^{8i+1}) + c^{8i+3} H(z^{8i+3}) + c^{8i+5} H(z^{8i+5}) + c^{8i+7} H(z^{8i+7})$

- Free memory environment [desktop/server], not focused on memory minimization
- Step 5 benefits from the vectorization in the same way as square-root

Ratios


Ratios for arithmetic operations
Ratio op/mul
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0.27
0.16
0.47
0.14
1.44
1.36
1.17
0.92
squaring
square root
multi-squaring
quadratic solver
$\mathbb{F}_{2^{233}}$
$\mathbb{F}_{2^{409}}$

Outline of the talk

1 Introduction

2 Algorithms and implementation
- Binary field arithmetic
- Elliptic curves arithmetic
- Scalar multiplication

3 Results

Elliptic curves arithmetic

- Set of curves used in this work:
  - NIST random binary elliptic curves: **B-233, B-409**
  - NIST Koblitz curves: **K-233, K-409**
  - Binary Edwards elliptic curve: **curve2251**
    $y^2 + xy = x^3 + (z^{13} + z^9 + z^8 + z^7 + z^2 + z + 1)$

Op.	LD	Kim&Kim *	Exp. result in $\mathbb{F}_{2^{233}}$
Point Doubling	4M, 5S	4M, 5S, -2Red	5.5M
Point Addition	8M, 5S	8M, 4S, -3Red	9M

* [Kim et al., IACR ePrint 07']

Op.	Theoretical cost	Exp. result in $\mathbb{F}_{2^{233}}$
Point Halving	1M, 1QS, 1SQRT	3.3M

Elliptic curves arithmetic

- Set of curves used in this work:
  - NIST random binary elliptic curves: **B-233, B-409**
  - NIST Koblitz curves: **K-233, K-409**
  - Binary Edwards elliptic curve: **curve2251**
    $y^2 + xy = x^3 + (z^{13} + z^9 + z^8 + z^7 + z^2 + z + 1)$

Op.	LD	Kim&Kim *	Exp. result in $\mathbb{F}_{2^{233}}$
Point Doubling	4M, 5S	4M, 5S, -2Red	5.5M
Point Addition	8M, 5S	8M, 4S, -3Red	9M

* [Kim et al., IACR ePrint 07']

Op.	Theoretical cost	Exp. result in $\mathbb{F}_{2^{233}}$
Point Halving	1M, 1QS, 1SQRT	3.3M

Outline of the talk

1 Introduction

2 Algorithms and implementation
- Binary field arithmetic
- Elliptic curves arithmetic
- Scalar multiplication

3 Results

Scalar multiplication

- Implementation methods and features:

Curve	Method	Parallel	SCP *
random	Double&add, Halve&add	OK	X
Koblitz	τ&add, τ^{-1}&add	OK	X
curve2251	Montgomery laddering	X	OK

* SCP = Side-Channel Protected

Scalar multiplication

- Implementation methods and features:

Curve	Method	Parallel	SCP *
random	Double&add, Halve&add	OK	X
Koblitz	τ&add, τ^{-1}&add	OK	X
curve2251	Montgomery laddering	X	OK

* SCP = Side-Channel Protected

- Today's focus = random curves

Sequential algorithm

Algorithm 2 Double-and-add scalar multiplication

Input: ω, k, $P \in E(\mathbb{F}_{2^m})$ of odd order r
Output: kP

Obtain the representation $\omega\text{NAF}(k) = \sum_{i=0}^{t} k_i 2^i$
Compute $P_i = iP$ for $i \in I = \{1, 3, \ldots, 2^{\omega-1} - 1\}$
$Q \leftarrow \mathcal{O}$
for $i = t$ **downto** 0 **do**
    $Q \leftarrow 2Q$
    if $k_i' > 0$ **then**
        $Q \leftarrow Q + P_{k_i}$
    else if $k_i' < 0$ **then**
        $Q \leftarrow Q - P_{-k_i}$
return Q

$$\text{cost} = \text{pre-comp} + \frac{t}{\omega+1}\text{PA} + t.\text{PD}$$

Sequential algorithm

Algorithm 3 Halve-and-add scalar multiplication [Fong et al., IEEE TC 04']

Input: ω, k, $P \in E(\mathbb{F}_{2^m})$ of odd order r
Output: kP

Perform scalar recoding: $k' = 2^t k \bmod r$ where $t = \lceil \log_2 r \rceil$
Obtain the representation $\omega\mathrm{NAF}(k')/2^t = \sum_{i=0}^{t} k'_i 2^{i-t}$
Initialize $Q_i \leftarrow \mathcal{O}$ for $i \in I = \{1, 3, \ldots, 2^{\omega-1} - 1\}$
for $i = t$ **downto** 0 **do**
    if $k'_i > 0$ **then**
        $Q_{k'_i} \leftarrow Q_{k'_i} + P$
    else if $k'_i < 0$ **then**
        $Q_{-k'_i} \leftarrow Q_{-k'_i} - P$
    $P \leftarrow P/2$
return $Q \leftarrow \sum_{i \in I} iQ_i$

$$\text{cost} = \frac{t}{\omega+1}\text{PA} + t.\text{PH} + \text{post-comp}$$

Parallel formulation

- Formula for parallel implementation on random binary curves:

$$kP = (k'_t 2^{t-n} + \cdots + k'_n)P + (k'_{n-1}2^{-1} + \cdots + k'_0 2^{-n})P$$

- In other words:

$$kP = \sum_{i=n}^{t} k'_i 2^{i-n} P + \sum_{i=0}^{n-1} k'_i 2^{-n+i} P$$

Parallel algorithm

Algorithm 4 Double-and-add, halve-and-add scalar multiplication: parallel

Input: ω, scalar k, $P \in E(\mathbb{F}_{2^m})$ of odd order r, constant $n \approx \frac{t}{2}$
Output: kP

Parallel algorithm

Algorithm 5 Double-and-add, halve-and-add scalar multiplication: parallel

Input: ω, scalar k, $P \in E(\mathbb{F}_{2^m})$ of odd order r, constant $n \approx \frac{t}{2}$
Output: kP

1: Compute $P_i = iP$ for $i \in I = \{1, 3, \ldots, 2^{\omega-1} - 1\}$
2: $Q_0 \leftarrow \mathcal{O}$
{Barrier}

3: Recode: $k' = 2^n k \bmod r$ and obtain rep $\omega\mathrm{NAF}(k')/2^n = \sum_{i=0}^{t} k'_i 2^{i-n}$
4: Initialize $Q_i \leftarrow \mathcal{O}$ for $i \in I$

Parallel algorithm

Algorithm 6 Double-and-add, halve-and-add scalar multiplication: parallel

Input: ω, scalar k, $P \in E(\mathbb{F}_{2^m})$ of odd order r, constant $n \approx \frac{t}{2}$
Output: kP

Compute $P_i = iP$ for $i \in I = \{1, 3, \ldots, 2^{\omega-1} - 1\}$
$Q_0 \leftarrow \mathcal{O}$

Recode: $k' = 2^n k \bmod r$ and obtain rep $\omega\mathrm{NAF}(k')/2^n = \sum_{i=0}^{t} k'_i 2^{i-n}$
Initialize $Q_i \leftarrow \mathcal{O}$ for $i \in I$

{Barrier}

```
for i = t downto n do
    Q_0 ← 2Q_0
    if k'_i > 0 then
        Q_0 ← Q_0 + P_{k'_i}
    else if k'_i < 0 then
        Q_0 ← Q_0 − P_{−k'_i}
```

Parallel algorithm

Algorithm 7 Double-and-add, halve-and-add scalar multiplication: parallel

Input: ω, scalar k, $P \in E(\mathbb{F}_{2^m})$ of odd order r, constant $n \approx \frac{t}{2}$

Output: kP

1: Compute $P_i = iP$ for $i \in I = \{1, 3, \ldots, 2^{\omega-1} - 1\}$
2: $Q_0 \leftarrow \mathcal{O}$
{Barrier}
5: **for** $i = t$ **downto** n **do**
6: $Q_0 \leftarrow 2Q_0$
7: **if** $k'_i > 0$ **then**
8: $Q_0 \leftarrow Q_0 + P_{k'_i}$
9: **else if** $k'_i < 0$ **then**
10: $Q_0 \leftarrow Q_0 - P_{-k'_i}$

3: Recode: $k' = 2^n k \bmod r$ and obtain rep $\omega\mathrm{NAF}(k')/2^n = \sum_{i=0}^{t} k'_i 2^{i-n}$
4: Initialize $Q_i \leftarrow \mathcal{O}$ for $i \in I$

Parallel algorithm

Algorithm 8 Double-and-add, halve-and-add scalar multiplication: parallel

Input: ω, scalar k, $P \in E(\mathbb{F}_{2^m})$ of odd order r, constant $n \approx \frac{t}{2}$

Output: kP

Compute $P_i = iP$ for $i \in I = \{1, 3, \ldots, 2^{\omega-1} - 1\}$
$Q_0 \leftarrow \mathcal{O}$
{Barrier}
for $i = t$ **downto** n **do**
  $Q_0 \leftarrow 2Q_0$
  if $k'_i > 0$ **then**
    $Q_0 \leftarrow Q_0 + P_{k'_i}$
  else if $k'_i < 0$ **then**
    $Q_0 \leftarrow Q_0 - P_{-k'_i}$
{Barrier}

Recode: $k' = 2^n k \bmod r$ and obtain rep $\omega\mathrm{NAF}(k')/2^n = \sum_{i=0}^{t} k'_i 2^{i-n}$
Initialize $Q_i \leftarrow \mathcal{O}$ for $i \in I$
for $i = n - 1$ **downto** 0 **do**
  $P \leftarrow P/2$
  if $k'_i > 0$ **then**
    $Q_{k'_i} \leftarrow Q_{k'_i} + P$
  else if $k'_i < 0$ **then**
    $Q_{-k'_i} \leftarrow Q_{-k'_i} - P$

Parallel algorithm

Algorithm 9 Double-and-add, halve-and-add scalar multiplication: parallel

Input: ω, scalar k, $P \in E(\mathbb{F}_{2^m})$ of odd order r, constant $n \approx \frac{t}{2}$
Output: kP

Compute $P_i = iP$ for $i \in I = \{1, 3, \ldots, 2^{\omega-1} - 1\}$	Recode: $k' = 2^n k \bmod r$ and obtain rep $\omega\mathrm{NAF}(k')/2^n = \sum_{i=0}^{t} k'_i 2^{i-n}$
$Q_0 \leftarrow \mathcal{O}$	Initialize $Q_i \leftarrow \mathcal{O}$ for $i \in I$
{Barrier}	
for $i = t$ **downto** n **do**	**for** $i = n - 1$ **downto** 0 **do**
$Q_0 \leftarrow 2Q_0$	$P \leftarrow P/2$
if $k'_i > 0$ **then**	**if** $k'_i > 0$ **then**
$Q_0 \leftarrow Q_0 + P_{k'_i}$	$Q_{k'_i} \leftarrow Q_{k'_i} + P$
else if $k'_i < 0$ **then**	**else if** $k'_i < 0$ **then**
$Q_0 \leftarrow Q_0 - P_{-k'_i}$	$Q_{-k'_i} \leftarrow Q_{-k'_i} - P$
{Barrier}	

return $Q \leftarrow Q_0 + \sum_{i \in I} iQ_i$

Outline of the talk

1 Introduction

2 Algorithms and implementation
- Binary field arithmetic
- Elliptic curves arithmetic
- Scalar multiplication

3 Results

Benchmark environment

- Timings validated with SUPERCOP ["turbo" mode disabled] from eBACS: `http://bench.cr.yp.to`
- Intel *Westmere* and *Sandy Bridge* families [respectively Core i5-660 and Core i7-2600K]
- Random scalar and unknown point scenario

Benchmark

- Timings in 10^3 clock cycles, (SC)=Single-Core, (MC)=Multi-Core

- 112-bit security level

Implementation	System	Finite Field	Westmere	Sandy Bridge
NIST - K-233 (SC)	τ&add (5τ-NAF)	$\mathbb{F}_{2^{233}}$	89	67.8
NIST - B-233 (SC)	Halve&add (4-NAF)	$\mathbb{F}_{2^{233}}$	182	157
NIST - K-233 (MC)	(τ|τ)&add (5τ-NAF)	$\mathbb{F}_{2^{233}}$	58	46.5
NIST - B-233 (MC)	(Dbl,Halve)&add (4-NAF)	$\mathbb{F}_{2^{233}}$	116	100

- 128-bit security level

Implementation	System	Finite Field	Westmere	Sandy Bridge
curve2251 (SC)	Montgomery	$\mathbb{F}_{2^{251}}$	282	225

- 192-bit security level

Implementation	System	Finite Field	Westmere	Sandy Bridge
NIST - K-409 (SC)	τ&add (5τ-NAF)	$\mathbb{F}_{2^{409}}$	321	255.6
NIST - B-409 (SC)	Halve&add (4-NAF)	$\mathbb{F}_{2^{409}}$	705	557
NIST - K-409 (MC)	(τ|τ)&add (5τ-NAF)	$\mathbb{F}_{2^{409}}$	191	148.8
NIST - B-409 (MC)	(Dbl,Halve)&add (4-NAF)	$\mathbb{F}_{2^{409}}$	444	349

Benchmark

- Timings in 10^3 clock cycles, (SC)=Single-Core, (MC)=Multi-Core

- 112-bit security level

Implementation	System	Finite Field	Westmere	Sandy Bridge
NIST - K-233 (SC)	τ&add (5τ-NAF)	$\mathbb{F}_{2^{233}}$	89	67.8
NIST - B-233 (SC)	Halve&add (4-NAF)	$\mathbb{F}_{2^{233}}$	182	157
NIST - K-233 (MC)	$(\tau\|\tau)$&add (5τ-NAF)	$\mathbb{F}_{2^{233}}$	58	46.5 (x1.46)
NIST - B-233 (MC)	(Dbl,Halve)&add (4-NAF)	$\mathbb{F}_{2^{233}}$	116	100 (x1.57)

- 128-bit security level

Implementation	System	Finite Field	Westmere	Sandy Bridge
curve2251 (SC)	Montgomery	$\mathbb{F}_{2^{251}}$	282	225

- 192-bit security level

Implementation	System	Finite Field	Westmere	Sandy Bridge
NIST - K-409 (SC)	τ&add (5τ-NAF)	$\mathbb{F}_{2^{409}}$	321	255.6
NIST - B-409 (SC)	Halve&add (4-NAF)	$\mathbb{F}_{2^{409}}$	705	557
NIST - K-409 (MC)	$(\tau\|\tau)$&add (5τ-NAF)	$\mathbb{F}_{2^{409}}$	191	148.8 (x1.72)
NIST - B-409 (MC)	(Dbl,Halve)&add (4-NAF)	$\mathbb{F}_{2^{409}}$	444	349 (x1.6)

Comparison with the literature

128-bits security level. Timings validated with SUPERCOP except for (*)

Implementation	System	Finite Field	10^3 clock cycles
Bernstein (*)	curve2251 Core 2 Quad Q6600	binary field: $\mathbb{F}_{2^{251}}$	314.3
Galbraith, Lin, Scott	gls1271 Intel Xeon E5620 (WSM)	prime field: $\mathbb{F}_{(2^{127}-1)^2}$	278.3
This work	curve2251 Intel Xeon E5620 (WSM)	binary field: $\mathbb{F}_{2^{251}}$	263.1
Bernstein *et al.*	curve25519 Intel Xeon E5620 (WSM)	prime field: $\mathbb{F}_{2^{255}-19}$	226.9
This work	curve2251 Intel Core i7-2600K (SB)	binary field: $\mathbb{F}_{2^{251}}$	**225**
Bernstein *et al.*	curve25519 Intel Core i7-2600K (SB)	prime field: $\mathbb{F}_{2^{255}-19}$	193.8
Hu, Longa, Xu (*)	Jac128gls4 Intel Core i7-2600M (SB)	prime field: $\mathbb{F}_{(2^{128}-40557)^2}$	120

Concluding remarks

- This work achieved an almost 2 factor of acceleration for parallel algorithms. However in practice this factor is up to 1.72 in our best implementation.
- Future improvement in parallelization management could improve this factor.
- If the latency would be reduced to 3 clock cycles as the instruction for integer, binary would have great chance to win against prime fields.
- AMD Bulldozer release is coming: will AMD's carry-less multiplication instruction latency be lower?
- Last update: scalar multiplication has been computed in 148.7×10^3 clock cycles using NIST recommended elliptic curve K-283. Moreover parallelized version takes 95.5×10^3.

Thank you!