Efficient and Secure (H)ECC Scalar Multiplication with Twin Multipliers

T. Lange* and P. K. Mishra°.

* Ruhr Universität Bochum, Germany.

° Indian Statistical Institute, Kolkata, India.

Basis

1. SCA resistant Parallel Explicit Formula for Addition and Doubling of Divisors in the Jacobian of Hyperelliptic Curves of Genus 2 (T. Lange and P. K. Mishra, Preprint)

2. Pipelined Computation of Scalar Multiplication in Elliptic Curve Cryptosystems. (P. K. Mishra, CHES 2004)

Overview

» (H)ECC
» Scalar Multiplication
» SCA n SCA
» ECC: Pipelining.
» (H)ECC: Parallelization.
» Security
» Efficiency

Introduction

- A hyperelliptic curve C of genus g (g > 0) over K is

$$C: y^2 + h(x)y = f(x)$$

where h , f are in K[x], deg (h) <= g, f is monic of degree of 2g+1 and there are no “singular points”. Elliptic curves are hyperelliptic curves of genus 1.

- The points of EC in KxK form an additive abelian group.
- In HEC, the group is the group of divisor classes of the curve.
- (H)ECC are El Gamal type cryptosystems built over these group.
- Advantages:
  - **No subexponential time algorithm for (H)ECDLP for curves of small genus.**
  - **A lot of curves (and other parameters) to choose from.**

Cost of Field Operations

- Cost of Field operations:
  - **Among [a], [m], [s], [i]; [a] is the cheapest.**
  - **Over binary fields [s] is slightly costlier than [a], but much cheaper than [m].**
  - **In prime fields we take [m] = [s].**
  - **[i] = k [m], where k is between 3 and 8 for binary fields, between 30 and 50 for prime fields. [i] is costliest, but occurs less frequently**.

- Arithmetic in affine coordinates involves inversion. So, other coordinate systems have been proposed.

- We use:
  - **For fields of characteristic 2 : affine coordinates**
  - **For fields of odd characterisitc :**
    - **Jacobian for ECC,**
    - **Lange's "new" coordinates for HECC.**

Cost of Group Operations

- ECC (Jacobian Coordinates)
  - **Addition (ECADD): 8[m] + 3[s] = 11[m]**
  - **Doubling (ECDBL): 6[m] + 4[s] = 10[m]**

- HECC (Affine Coordinates)
  - **Addition (HCDBL): 1[i] + 21[m] + 3[s]**
  - **Doubling (HCDBL): 1[i] + 22[m] + 5[s]**

- HECC (Lange‘s new Coordinates)
  - **Addition (HCADD): 38[m] + 6[s] = 44[m]**
  - **Doubling (HCDBL): 37[m] + 4[4] = 41[m]**

Scalar Mutiplication

- Computationally the most dominant operation in (H)ECC.
- Generally computed by a series of doublings and additions.

The binary algorithm (L2R)

Input: Integer m $(m_{n-1}m_{n-2} \ldots m_0)_2$ and a point P

Output: mP

1. Let Q = P
2. For i = n-2 down to 0
   - Q = DBL(Q)
   - if m_i = 1 then Q = ADD(Q , P)
3. Return (Q)

–

SCA and SCA

- **Use of side-channel info like timing, power consumption and EM radiation traces**
- **Countermeasures against SPA-like Attacks:**
  - **Double and always add**
  - **Various addition chains**
  - **Unified Algorithms**
  - **Side Channel Atomicity**
- **Randomization is the main technique against DPA-like Attacks:**
  - **curve randomization**
  - **point randomization**
  - **scalar multiplier randomization.**
- **Most of these techniques are similar for ECC and HECC.**
- **We use the side-channel atomicity to resist SPA. Any countermeasure against DPA can be securely integrated to it.**

SCA and SCA

- SCA is the most recent and most economic countermeasure against SPA.
- Proposed by Chevallier-Mames, Ciet and Joye in 2002.
- It divides the ECADD and ECDBL into indistinguishable atomic blocks. Computation of a series of DBL and ADD looks like computation of a series of atomic blocks. No information about the operation being processed is leaked out.
- Overhead: only some inexpensive field operations like additions and subtractions.
- We use side-channel atomicity to shield our method against SPA. All standard countermeasure against DPA can be incorporated to it.

How does it look like?

ECADD in Atomic Blocks

ADD Algorithm

Input: $P = (T_x, T_y), P_i = (X_i, Y_i, Z_i)$

Output: $P + P_i = (X_{i+1}, Y_{i+1}, Z_{i+1})$.

Γ_1	$R_1 = T_8 \times T_8 \ (Z_i^2)$ $*$ $*$ $*$	Γ_7	$R_2 = R_2 \times R_4 \ (-U_1W^2)$ $R_5 = R_2 + R_2 \ (-2U_1W^2)$ $*$ $*$
Γ_2	$R_2 = T_x \times R_1 \ (U_1)$ $*$ $R_2 = -R_2 \ (-U_1)$ $*$	Γ_8	$R_1 = R_4 \times R_1 \ (W^3)$ $R_1 = R_1 + R_5 \ (W^3 - 2U_1W^2)$ $R_3 = -R_3 \ (-S_1)$ $R_5 = R_3 + T_7 \ (S_2 - S_1 = -R)$
Γ_3	$R_3 = T_y \times T_8 \ (YZ_i)$ $*$ $*$ $*$	Γ_9	$T_6 = R_5 \times R_5 \ (R^2)$ $T_6 = T_6 + R_1 \ (\underline{X_{i+1}})$ $R_2 = T_6 + R_2 \ (X_{i+1} - U_1W^2)$
Γ_4	$R_3 = R_3 \times R_1 \ (S_1)$ $R_1 = R_2 + T_6 \ (-W)$ $R_1 = -R_1 \ (W)$ $*$	Γ_{10}	$R_2 = R_5 \times R_2 \ (-R(X_{i+1} - U_1W^2))$ $*$ $*$
Γ_5 Γ_5	$T_8 = R_1 \times T_8 \ (Z_{i+1})$ $T_8 = R_1 \times T_8 \ (\underline{Z_{i+1}})$ $*$ $*$ $*$	Γ_{11} Γ_{11}	$T_7 = R_3 \times R_4 \ (S_1W^2)$ $T_7 = R_3 \times R_4 \ (-S_1W^2)$ $T_7 = T_7 + R_2 \ (\underline{Y_{i+1}})$ $*$ $*$
Γ_6	$R_4 = R_1 \times R_1 \ (W^2)$ $*$ $*$ $*$		

Algorithm HCDBL

Input: $D = (U_0, U_1, V_0, V_1, Z_1, Z_2, z_1, z_2)$

Out: $2D = (U_0', U_1', V_0', V_1', Z_1', Z_2', z_1', z_2')$

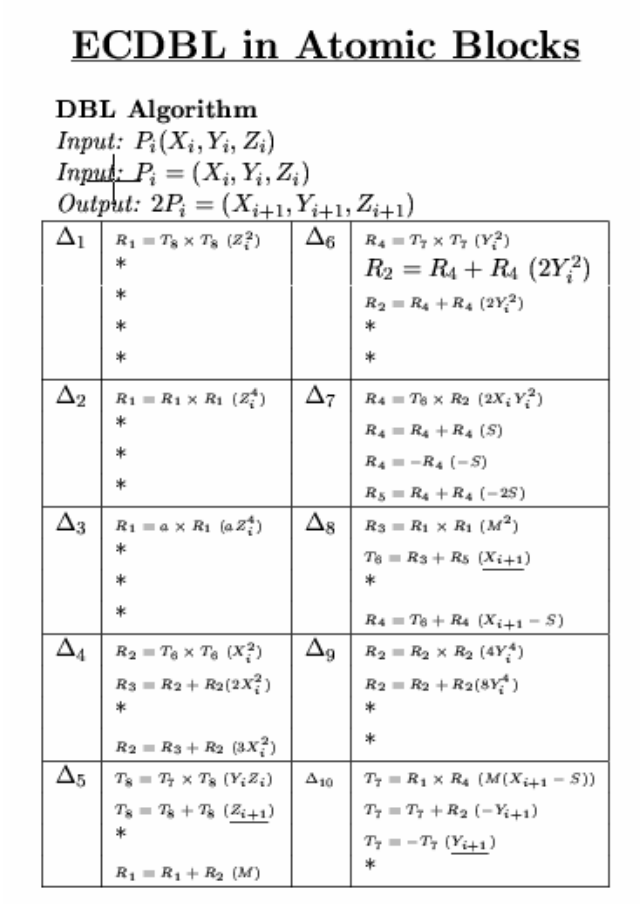
Init: $T_1 = U_0, T_2 = U_1, T_3 = V_0, T_4 = V_1,$
$T_5 = Z_1, T_6 = Z_2, T_7 = z_1, T_8 = z_2$

Γ_1	$T_9 = T_4 * T_4$ $*$ $*$ $*$	Γ_2	$T_{10} = T_2 * T_4$ $*$ $T_{10} = -T_{10}$ $*$
Γ_3	$T_{11} = T_3 * T_7$ $T_{11} = T_{11} + T_{10}$ $T_4 = -T_4$ $*$	Γ_4	$T_{12} = T_9 * T_1$ $*$ $*$ $*$
Γ_5	$T_{10} = T_3 * T_{11}$ $T_{10} = T_{10} + T_{12}$ $T_9 = -T_9$ $*$	Γ_6	$T_{13} = T_1 * T_7$ $*$ $T_{13} = -T_{13}$ $*$
Γ_7	$T_{12} = T_2 * T_2$ $T_{14} = T_{12} + T_{13}$ $T_{13} = -T_{13}$ $T_{14} = T_{14} + T_{14}$	Γ_8	$T_{15} = T_7 * T_7$ $*$ $*$ $*$
Γ_9	$T_{16} = T_{15} * f3$ $T_{16} = T_{16} + T_{12}$ $*$ $T_{12} = T_{13} + T_{13}$	Γ_{10}	$T_6 = T_6 * T_{10}$ $*$ $*$ $*$
Γ_{11}	$T_{15} = T_{15} * T_7$ $T_{14} = T_{14} + T_{16}$ $T_{16} = -T_{16}$ $T_{12} = T_{12} + T_{12}$	Γ_{12}	$T_6 = T_6 * T_7$ $*$ $*$ $*$
Γ_{13}	$T_{15} = T_{15} * f2$ $T_{12} = T_{12} + T_{16}$ $*$ $*$	Γ_{14}	$T_{14} = T_8 * T_{14}$ $*$ $T_{14} = -T_{14}$ $*$
Γ_{15}	$T_{12} = T_{12} * T_2$ $T_{12} = T_{12} + T_{15}$ $T_{15} = -T_6$ $T_2 = T_2 + 1$	Γ_{16}	$T_{16} = T_{14} * T_4$ $*$ $*$ $*$

ECC: Pipelining(1)

- Assumptions for Pipelining
  - One basic observation: in the scalar multiplication algorithm the EC-operations can be cascaded if adequate hardware support available.
  - One more multiplier will do the trick.
  - Both operations in the pipeline get their i/p and write back their o/p to the three fixed locations: say T_6, T_7, T_8. Fortunately, no conflicts.
  - The base point in affine is stored at a fixed location, say, T_x, T_y.
  - Both PS have 5 locations each to store their intermediate variables. Needs more memory .

ECDBL in Atomic Blocks



ECDBL in Atomic Blocks

DBL Algorithm

Input: $P_i(X_i, Y_i, Z_i)$

Input: $P_i = (X_i, Y_i, Z_i)$

Output: $2P_i = (X_{i+1}, Y_{i+1}, Z_{i+1})$

Δ_1	$R_1 = T_8 \times T_8 \ (Z_i^2)$ * * * *	Δ_6	$R_4 = T_7 \times T_7 \ (Y_i^2)$ $R_2 = R_4 + R_4 \ (2Y_i^2)$ $R_2 = R_4 + R_4 \ (2Y_i^2)$ * *
Δ_2	$R_1 = R_1 \times R_1 \ (Z_i^4)$ * * *	Δ_7	$R_4 = T_8 \times R_2 \ (2X_i Y_i^2)$ $R_4 = R_4 + R_4 \ (S)$ $R_4 = -R_4 \ (-S)$ $R_5 = R_4 + R_4 \ (-2S)$
Δ_3	$R_1 = a \times R_1 \ (aZ_i^4)$ * * *	Δ_8	$R_3 = R_1 \times R_1 \ (M^2)$ $T_8 = R_3 + R_5 \ (X_{i+1})$ * $R_4 = T_8 + R_4 \ (X_{i+1} - S)$
Δ_4	$R_2 = T_8 \times T_8 \ (X_i^2)$ $R_3 = R_2 + R_2 (2X_i^2)$ * $R_2 = R_3 + R_2 \ (3X_i^2)$	Δ_9	$R_2 = R_2 \times R_2 \ (4Y_i^4)$ $R_2 = R_2 + R_2 (8Y_i^4)$ * *
Δ_5	$T_8 = T_7 \times T_8 \ (Y_i Z_i)$ $T_8 = T_8 + T_8 \ (Z_{i+1})$ * $R_1 = R_1 + R_2 \ (M)$	Δ_{10}	$T_7 = R_1 \times R_4 \ (M(X_{i+1} - S))$ $T_7 = T_7 + R_2 \ (-Y_{i+1})$ $T_7 = -T_7 \ (Y_{i+1})$ *

- The atomic blocks Δ_1, Δ_2, Δ_3 can be computed with the input Z_i only.
- Input X_i is needed by ECDBL at block Δ_4 and thereafter.
- The block Δ_5 needs the input Y_i as well. But Δ_5 produces the output Z_{i+1}. So, the next operation can begin after ECDBL completes Δ_5.
- The atomic block Δ_8 produces the output X_{i+1}.
- The block Δ_{10} produces the output Y_{i+1} and the process terminates.

ECADD in Atomic Blocks

ECADD in Atomic Blocks

ADD Algorithm

Input: $P = (T_x, T_y), P_i = (X_i, Y_i, Z_i)$

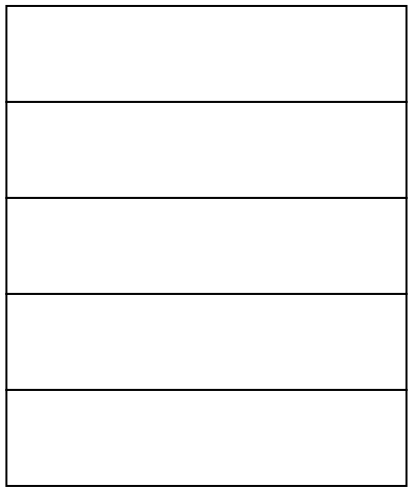
Output: $P + P_i = (X_{i+1}, Y_{i+1}, Z_{i+1})$.

Γ_1	$R_1 = T_8 \times T_8 \ (Z_i^2)$ * * *	Γ_7	$R_2 = R_2 \times R_4 \ (-U_1 W^2)$ $R_5 = R_2 + R_2 \ (-2U_1 W^2)$ * *
Γ_2	$R_2 = T_x \times R_1 \ (U_1)$ * $R_2 = -R_2 \ (-U_1)$ *	Γ_8	$R_1 = R_4 \times R_1 \ (W^3)$ $R_1 = R_1 + R_5 \ (W^3 - 2U_1 W^2)$ $R_3 = -R_3 \ (-S_1)$ $R_5 = R_3 + T_7 \ (S_2 - S_1 = -R)$
Γ_3	$R_3 = T_y \times T_8 \ (Y Z_i)$ * * *	Γ_9	$T_8 = R_5 \times R_5 \ (R^2)$ $T_8 = T_8 + R_1 \ (\underline{X_{i+1}})$ $R_2 = T_8 + R_2 \ (X_{i+1} - U_1 W^2)$
Γ_4	$R_3 = R_3 \times R_1 \ (S_1)$ $R_1 = R_2 + T_8 \ (-W)$ $R_1 = -R_1 \ (W)$ *	Γ_{10}	$R_2 = R_5 \times R_2 \ (-R(X_{i+1} - U_1 W^2))$ * *
Γ_5 Γ_5	$T_8 = R_1 \times T_8 \ (\underline{Z_{i+1}})$ $T_8 = R_1 \times T_8 \ (\underline{Z_{i+1}})$ * * *	Γ_{11} Γ_{11}	$T_7 = R_3 \times R_4 \ (S_1 W^2)$ $T_7 = R_3 \times R_4 \ (-S_1 W^2)$ $T_7 = T_7 + R_2 \ (\underline{Y_{i+1}})$ * *
Γ_6	$R_4 = R_1 \times R_1 \ (W^2)$ * * *		

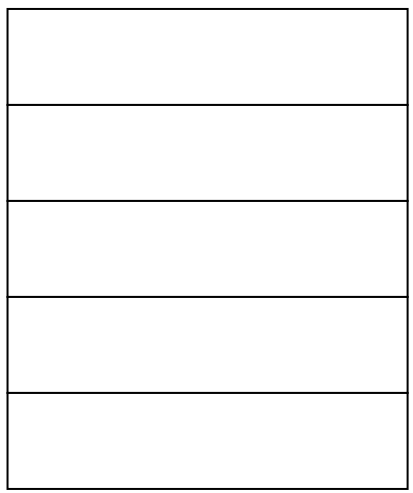
- The atomic blocks Γ_1, Γ_2, Γ_3 can be computed with the input Z_i only.
- Input X_i is needed by ECADD at block Γ_4 and thereafter.
- The block Γ_5 produces the output Z_{i+1}. So, the next operation can begin after ECADD completes Γ_5.
- The input Y_i is not required till the atomic block Γ_8.
- The block Γ_9 produces the output X_{i+1} and Γ_{11} produces Y_{i+1} and the process terminates.

Pipelining

1

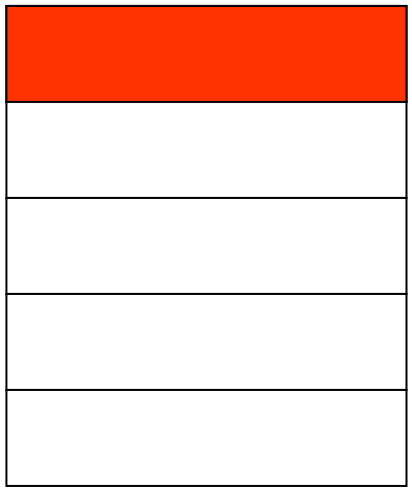


PS1

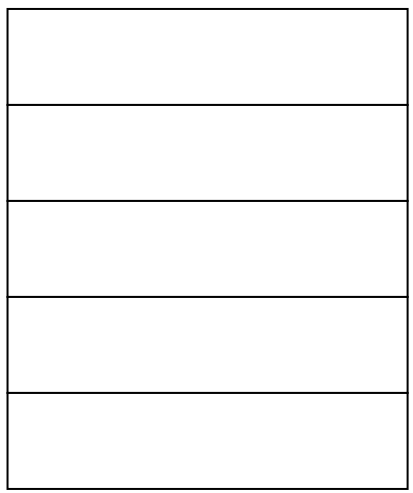


PS2

Pipelining: DBL-DBL



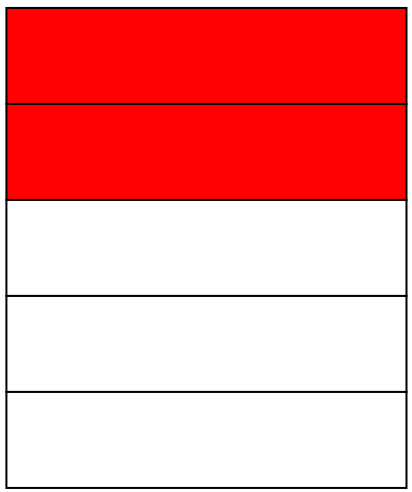
1



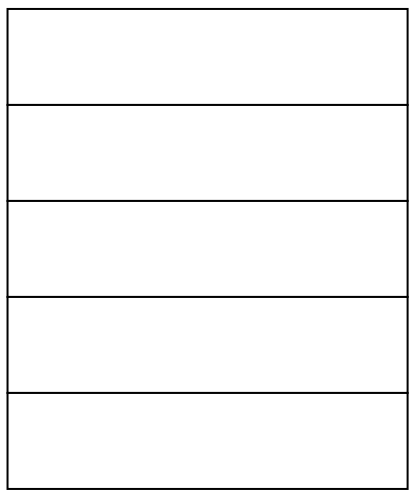
PS1

PS2

Pipelining: DBL-DBL



PS1

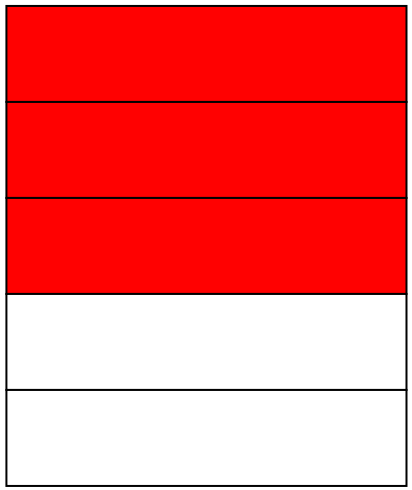


PS2

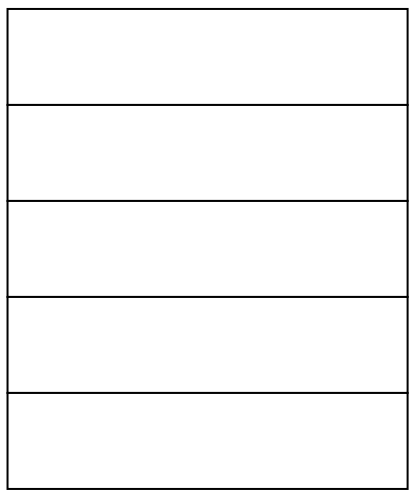
2

Pipelining: DBL-DBL

3



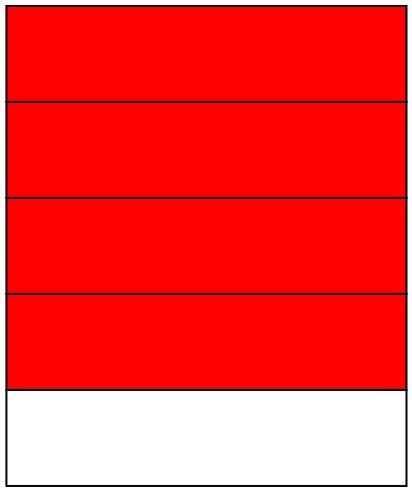
PS1



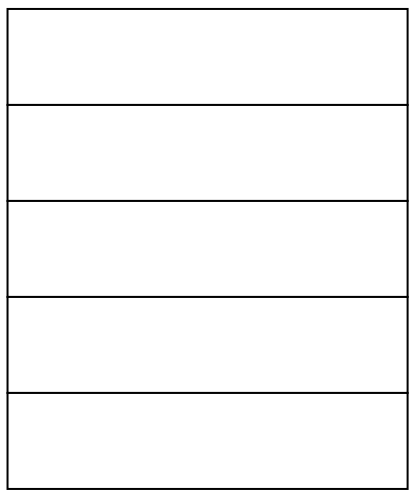
PS2

Pipelining: DBL-DBL

4



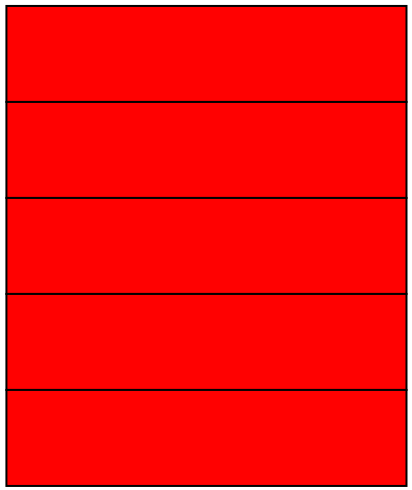
PS1



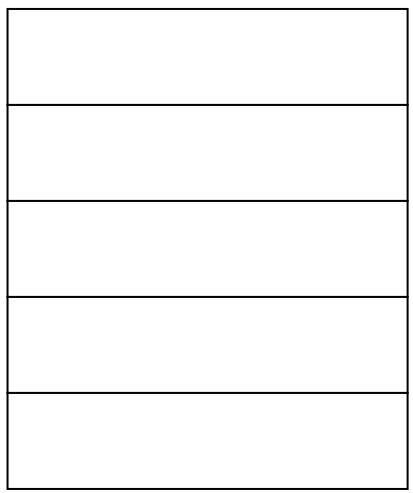
PS2

Pipelining: DBL-DBL

5



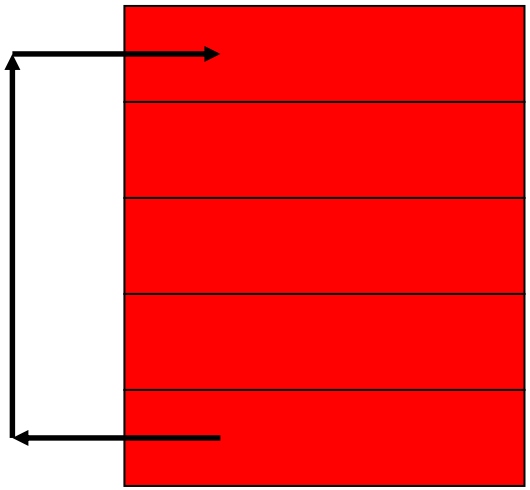
PS1



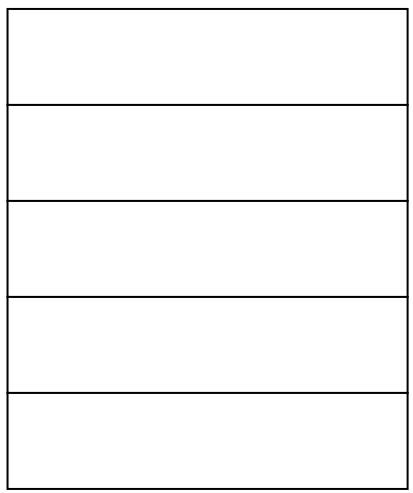
PS2

Pipelining: DBL-DBL

5



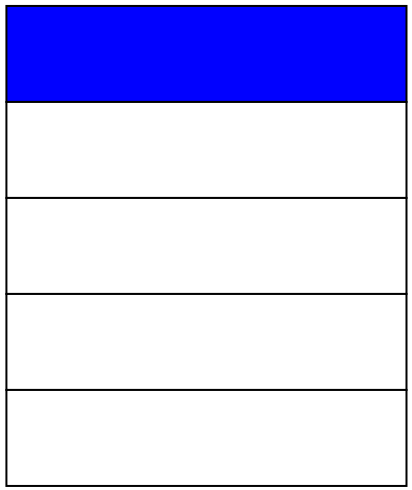
PS1



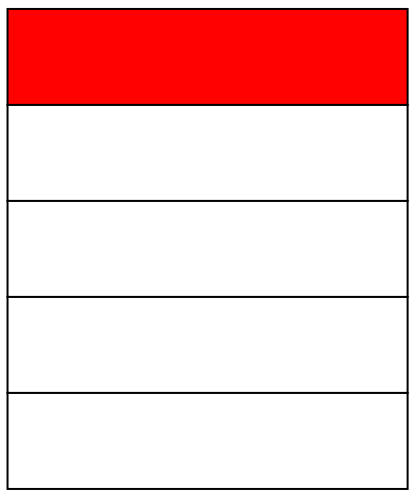
PS2

Pipelining: DBL-DBL

6

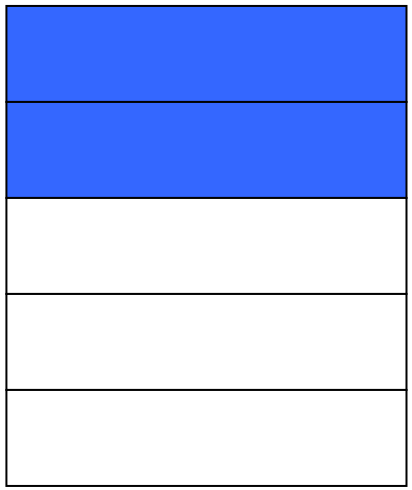


PS1

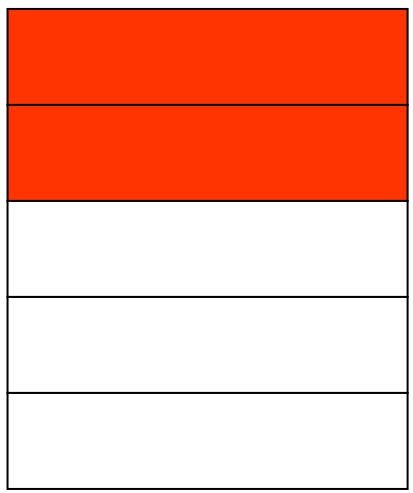


PS2

Pipelining: DBL-DBL



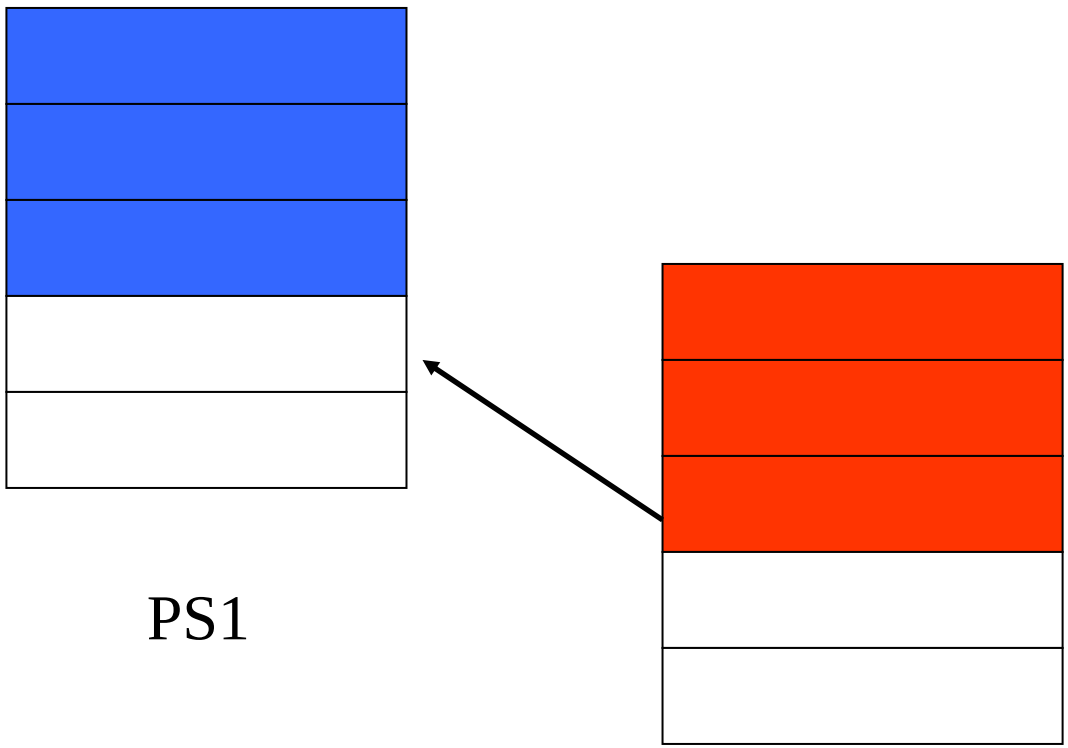
PS1



PS2

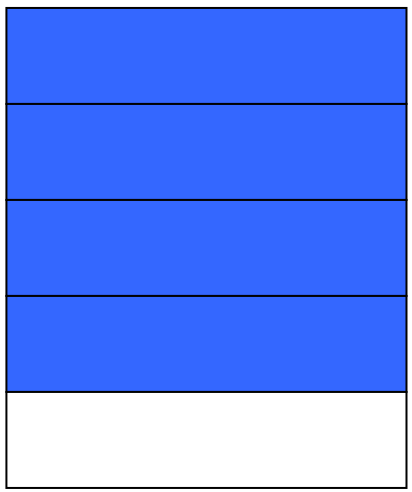
Pipelining: DBL-DBL

8

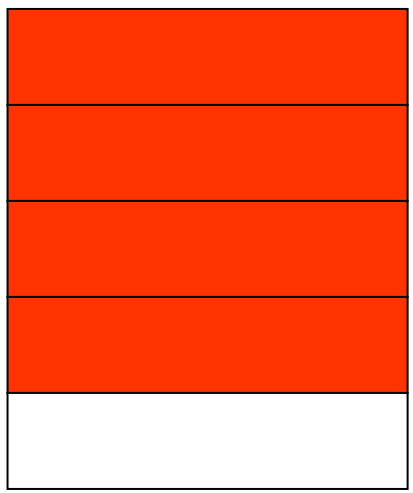


Pipelining: DBL-DBL

9



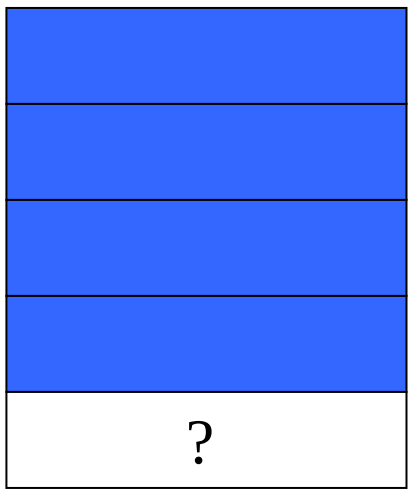
PS1



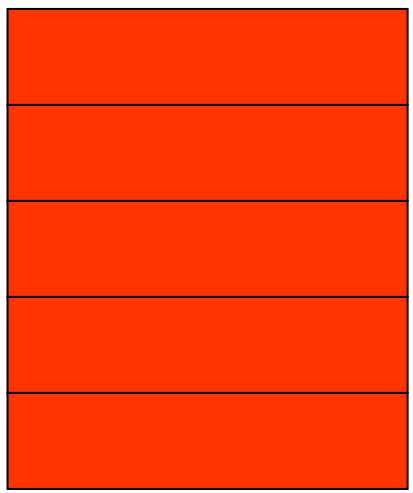
PS2

Pipelining: DBL-DBL

10

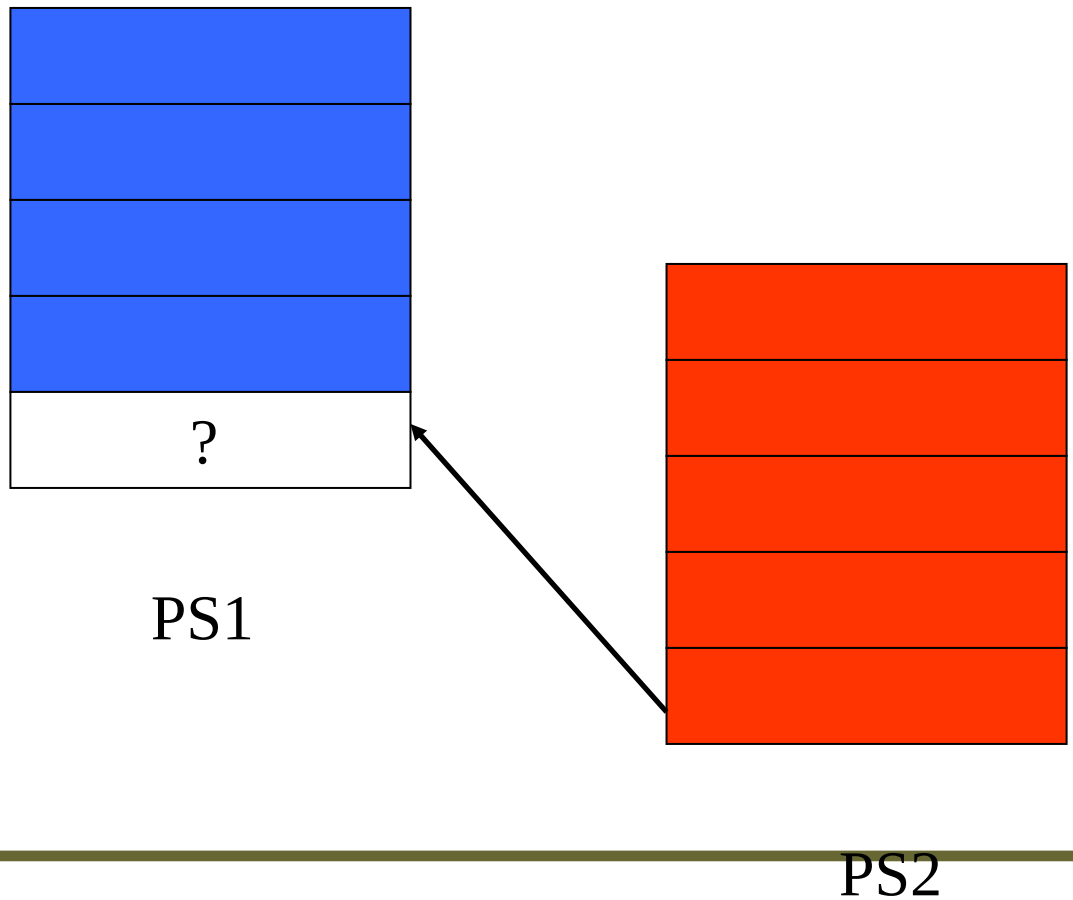


PS1

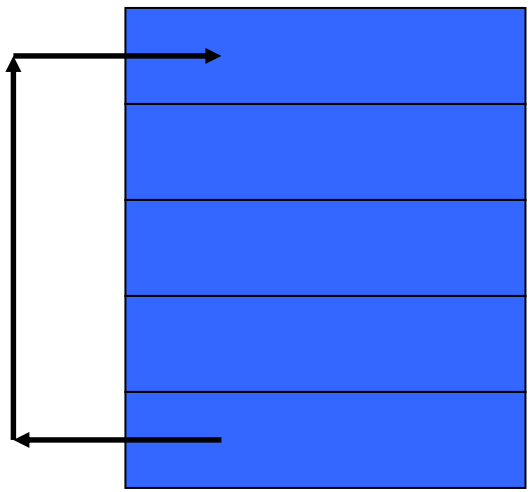


PS2

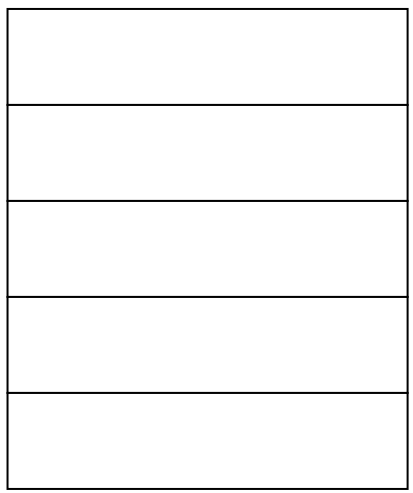
Pipelining: DBL-DBL



Pipelining: DBL-DBL

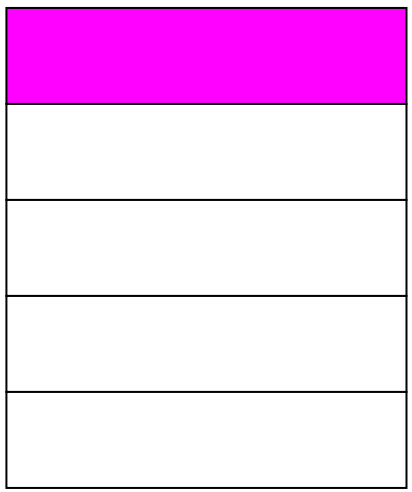


PS1

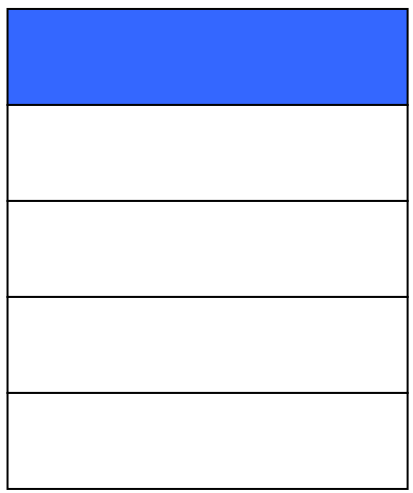


PS2

Pipelining: DBL-DBL



PS1



PS2

Pipelining: Other Scenarios

Pipelining

	DBL-DBL		DBL-ADD		ADD-DBL	
Time	PS1	PS2	PS1	PS2	PS1	PS2
k	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$k+1$	$\Delta_1^{(i)}$	-	$\Delta_1^{(i)}$	-	$\Gamma_1^{(i)}$	-
$k+2$	$\Delta_2^{(i)}$	-	$\Delta_2^{(i)}$	-	$\Gamma_2^{(i)}$	-
$k+3$	$\Delta_3^{(i)}$	-	$\Delta_3^{(i)}$	-	$\Gamma_3^{(i)}$	-
$k+4$	$\Delta_4^{(i)}$	-	$\Delta_4^{(i)}$	-	$\Gamma_4^{(i)}$	-
$k+5$	$\Delta_5^{(i)}$	-	$\Delta_5^{(i)}$	-	$\Gamma_5^{(i)}$	-
$k+6$	$\Delta_1^{(i+1)}$	$\Delta_6^{(i)}$	$\Gamma_1^{(i+1)}$	$\Delta_6^{(i)}$	$\Delta_1^{(i+1)}$	$\Gamma_6^{(i)}$
$k+7$	$\Delta_2^{(i+1)}$	$\Delta_7^{(i)}$	$\Gamma_2^{(i+1)}$	$\Delta_7^{(i)}$	$\Delta_2^{(i+1)}$	$\Gamma_7^{(i)}$
$k+8$	$\Delta_3^{(i+1)}$	$\Delta_8^{(i)}$	$\Gamma_3^{(i+1)}$	$\Delta_8^{(i)}$	$\Delta_3^{(i+1)}$	$\Gamma_8^{(i)}$
$k+9$	$\Delta_4^{(i+1)}$	$\Delta_9^{(i)}$	$\Gamma_4^{(i+1)}$	$\Delta_9^{(i)}$	*	$\Gamma_9^{(i)}$
$k+10$	*	$\Delta_{10}^{(i)}$	$\Gamma_5^{(i+1)}$	$\Delta_{10}^{(i)}$	$\Delta_4^{(i+1)}$	$\Gamma_{10}^{(i)}$
$k+11$	$\Delta_5^{(i+1)}$	*	$\vdots$	$\Gamma_6^{(i+1)}$	*	$\Gamma_{11}^{(i)}$
$k+12$	$\vdots$	$\Delta_6^{(i+1)}$	$\vdots$	$\Gamma_7^{(i+1)}$	$\Delta_5^{(i+1)}$	*

Pipelining: Security

- The security of the scheme against SPA comes from the fact that it uses side channel atomicity.
- The DPA can be resisted by using Curve Randomization Countermeasure.
- Any other DPA countermeasure which works with affine representation of the base point can be integrated to the scheme.

- Let m be of n bits with hamming weight h. Then the binary algorithm needs $n-1$ ECDBL and $h-1$ ECADD.
- Pipelining needs 7 units of time for the first operation and 6 for each subsequent one.
- Hence time required is $7+6(n+h-3) = 6(n+h)-11$. For binary algorithm $h=n/2$, for NAF $h=n/3$ on average. Hence time required $9n$ and $8n$ respectively.
- Some pipestages are being wasted.
- Comparison for $n=160$ is given below.

Algorithm	Binary	NAF	w-NAF $(w = 4)$
Sequential	2477	2177	1893
Pipelined	1438	1278	1152

HECC Parallelization: Introduction

- HECC is now implemented via explicit formulae
- The most efficient such formulae for most general curves of genus 2 are proposed by Lange.
- Our task: to introduce the concept of side-channel atomicity into these formulae. Also, we want our formulae to be such that it can be easily run in parallel if sufficient hardware are available.
- Task is very much implementation dependent. We restrict to the most general situation.

HECC Parallelization: Introduction

- **Equation for curves of genus 2:**

$$y^2+(h_2x^2+h_1x+h_0)y = x^5+f_4x^4+f_3x^3+f_2x^2+f_1x+f_0 .$$

- **If the charcteristic of the field is not 5, f_4 can be made 0. Also, h_2 can be always made 0 or 1.**

- **As in binary fields the I/M ratio is between 8 to 10, one prefers affine arithmetic.**

- **Affine arithmetic involves inversion. We can not divide the group operations into smaller atomic blocks. Hence, we make each operation one block in even characteristic.**

- **In odd characteristic we divided the HCADD and HCDBL into smaller atomic blocks.**

HECC: Even Characteristic

- **Cost in even characterisitc:**
  - **HCADD: 1[i]+22[m]+3[s] HCDBL: 1[i]+22[m]+5[s]**
- **Inversions must occur at the same places in both HCADD and HCDBL. Besides all other operations must match. Addition of dummy [m] and [s] should be minimum.**
- **Number of operations before inversion:**
  - **HCADD: 9[m]+1[s] HCDBL: 11[m]+2[s]**
- **One [m] in HCDBL can be brought below the inversion. Hence 1 dummy [m] and 1[s] in HCADD are inevitable.**
- **After the inversion : HCADD: 13[m]+2[s] HCDBL: 12[m]+3[s]**
- **Hence 1 dummy [s] in HCADD and 1 dummy [m] in HCDBL are inevitable.**
- **Overhead: 2 dummy [m] and 2 dummy [s] besides some dummy additions.**

HECC: Even Characteristic

	Algorithm HCADD	**Algorithm HCDBL**
Input:	$D_1 = (u_{10}, u_{11}, v_{10}, v_{11})$ $D_2 = (u_{20}, u_{21}, v_{20}, v_{21})$	$D = (u_0, u_1, v_0, v_1)$
Output:	$D_1 + D_2 = (u'_0, u'_1, v'_0, v'_1)$	$2D = (u'_0, u'_1, v'_0, v'_1)$
Init:	$T_1 \leftarrow u_{10}, T_2 \leftarrow u_{11}$ $T_3 \leftarrow v_{10}, T_4 \leftarrow v_{11}, T_5 \leftarrow u_{20}$ $T_6 \leftarrow u_{21}, T_7 \leftarrow v_{20}, T_8 \leftarrow v_{21}$	$T_1 \leftarrow u_0, T_2 \leftarrow u_1$ $T_3 \leftarrow v_0, T_4 \leftarrow v_1$
1	$T_9 = T_2 + h_2 T_6$ (inv_1)	$T_5 = h_1 + T_2$ $(\tilde{v}_1 = inv_1)$
2	$T_{10} = T_1 + T_5$ (z_2)	$T_7 = h_0 + T_1$ $(\tilde{v}_0)$
3	$T_{11} = T_9 * T_9$	$T_6 = T_2 * T_2$ (w_1)
4	$*[s]$	$T_8 = T_5 * T_5$ (w_2)
5	$T_{13} = T_2 * T_9$	$T_9 = T_2 * T_5$ (w_3)
6	$T_{11} = T_1 * T_{11}$	$T_8 = T_1 * T_8$
7	$T_{12} = T_{10} + T_{13}$ (inv_0)	$T_9 = T_7 + T_9$ (inv_0)
8	$T_{13} = T_{10} * T_{12}$	$T_7 = T_7 * T_9$
9	$T_{10} = T_{11} + T_{13}$ (r)	$T_7 = T_8 + T_7$ (r)
10	$T_3 = T_3 + T_7$ (w_0)	$T_8 = f_3 + T_6$ (w_3)
11	$T_4 = T_4 + T_8$ (w_1)	$T_{10} = T_8 + h_2 T_4$ (k'_1)
12	$T_{11} = T_9 + T_{12}$	$T_8 = h_2 T_4 + T_8$
13	$T_{13} = T_3 + T_4$	$T_8 = f_2 + h_2 T_3$
14	$T_{14} = 1 + T_2$	$T_{11} = h_1 + T_4$
15	$T_3 = T_3 * T_{12}$ (w_2)	$T_8 = T_2 * T_8$
16	$T_4 = T_4 * T_9$ (w_3)	$T_{11} = T_4 * T_{11}$
17	$*[a]$	$T_8 = T_8 + T_{11}$
18	$*[a]$	$T_8 = T_6 + T_8$ (k'_0)
19	$T_{12} = T_{11} * T_{13}$	$T_{11} = T_8 * T_9$ (w_0)
20	$T_{15} = T_4 * T_{14}$	$T_{12} = T_{10} * T_5$ (w_1)
21	$T_{12} = T_3 + T_{12}$	$T_5 = T_5 + T_9$
22	$T_{12} = T_{12} + T_{15}$ (s'_1)	$T_8 = T_8 + T_{10}$
23	$T_4 = T_1 * T_4$	$T_5 = T_5 * T_8$
24	$T_4 = T_3 + T_4$ (s'_0)	$T_5 = T_5 + T_{11}$
25	$*[a]$	$T_{13} = 1 + T_2$
26	$T_3 = T_{10} * T_{12}$	$T_{13} = T_{12} * T_{13}$
27	$*[a]$	$T_5 = T_5 + T_{13}$ (s'_1)
28	$*[m]$	$T_8 = T_7 * T_5$
29	$T_3 = 1/T_3$ (w_1)	$T_8 = 1/T_8$ (w_1)

HECC: Parallelization.

- Let the two multipliers be M_1 and M_2.
- We propose a scheme in which the multipliers, the adder and the inverter are provided with operations in the same order for both the group operation HCADD and HCDBL. So both the operations become indistinguishable from the side-channel.
- For both th group operations the number of field operations is same, the order of the operations ia also same.
- This makes the scheme SPA resistant.
- In even characteristic as the inputs are in affine coordinates, curve randmomization can be adopted.

HECC: Parallelization.

Rnd	Operation	Op Code	Rnd	Operation	Op Code
1	+	1, 2	17	+	36
2	*	M_1: 3, M_2: 4	18	*	M_1: 34, M_2: 37
3	*	M_1: 5, M_2: 6	19	+	38
4	+	7	20	*	M_1: 39, M_2: 43
5	*	M_1: 8, M_2: -	21	+	40, 41, 42
6	+	9,10,11,12,13,14	22	*	M_1: 44, M_2: 46
7	*	M_1: 15, M_2: 16	23	+	45, 47, 48, 49
8	+	17, 18	24	*	M_1: 50, M_2: -
9	*	M_1: 19, M_2: 20	25	+	51, 52, 53
10	+	21, 22, 25	26	*	M_1: 54, M_2; 61
11	*	M_1: 23, M_2: 26	27	+	55, 56
12	+	24, 27	28	*	M_1: 57, M_2: -
13	*	M_1: 28, M_2: 31	29	+	58, 59, 60, 62
14	Inversion 29				
15	*	M_1: 30, M_2: 32	30	*	M_1: 63, M_2: -
16	*	M_1: 33, M_2: 35	31	+	64, 65, 66

Total number of rounds: 31, 1[i] + 16[m] + 14[a]

HECC: Odd Characteristic

- Cost of HCDBL: 34[m]+7[s] = 41[m]
- Cost of HCADD (mixed coord Affine+New = New): 38[m]+6[s] = 44[m]
- No inversion. So we divide them into smaller atomic blocks: each block containing one multiplication, two additions and one negation in the same order.
- Also we want parallelization. We design the blocks so that one even numbered and one odd numbered block can be executed in parallel. Conflicts should be avoided.
- Our Methodology:
  - **Split the explicit formula into three address codes**
  - **Identify the multiplications which can be executed in parallel**
  - **Attach addition operations to the multiplications to make one block each**
  - **Take care to avoid conflicts**.

HCDBL in Atomic Blocks

Algorithm HCDBL

Input: $D = (U_0, U_1, V_0, V_1, Z_1, Z_2, z_1, z_2)$
Out: $2D = (U_0', U_1', V_0', V_1', Z_1', Z_2', z_1', z_2')$
Init: $T_1 = U_0, T_2 = U_1, T_3 = V_0, T_4 = V_1,$
$T_5 = Z_1, T_6 = Z_2, T_7 = z_1, T_8 = z_2$

Γ_1	$T_9 = T_4 * T_4$ * * *	Γ_2	$T_{10} = T_2 * T_4$ * $T_{10} = -T_{10}$ *
Γ_3	$T_{11} = T_3 * T_7$ $T_{11} = T_{11} + T_{10}$ $T_4 = -T_4$ *	Γ_4	$T_{12} = T_9 * T_1$ * * *
Γ_5	$T_{10} = T_3 * T_{11}$ $T_{10} = T_{10} + T_{12}$ $T_9 = -T_9$ *	Γ_6	$T_{13} = T_1 * T_7$ * $T_{13} = -T_{13}$ *
Γ_7	$T_{12} = T_2 * T_2$ $T_{14} = T_{12} + T_{13}$ $T_{13} = -T_{13}$ $T_{14} = T_{14} + T_{14}$	Γ_8	$T_{15} = T_7 * T_7$ * * *
Γ_9	$T_{16} = T_{15} * f3$ $T_{16} = T_{16} + T_{12}$ * $T_{12} = T_{13} + T_{13}$	Γ_{10}	$T_6 = T_6 * T_{10}$ * * *
Γ_{11}	$T_{15} = T_{15} * T_7$ $T_{14} = T_{14} + T_{16}$ $T_{16} = -T_{16}$ $T_{12} = T_{12} + T_{12}$	Γ_{12}	$T_6 = T_6 * T_7$ * * *
Γ_{13}	$T_{15} = T_{15} * f2$ $T_{12} = T_{12} + T_{16}$ * *	Γ_{14}	$T_{14} = T_8 * T_{14}$ * $T_{14} = -T_{14}$ *
Γ_{15}	$T_{12} = T_{12} * T_2$ $T_{12} = T_{12} + T_{15}$ $T_{15} = -T_6$ $T_2 = T_2 + 1$	Γ_{16}	$T_{16} = T_{14} * T_4$ * * *

Algorithm HCADD

Input: $D_1 = (U_{10}, U_{11}, V_{10}, V_{11}, 1, 1, 1, 1)$
$D_2 = (U_{20}, U_{21}, V_{20}, V_{21}, Z_{21}, Z_{22}, z_{21}, z_{22})$
Out: $D_1 + D_2 = (U_0', U_1', V_0', V_1', Z_1', Z_2', z_1', z_2')$
Init: $T_1 = U_{10}, T_2 = U_{11}, T_3 = V_{10}, T_4 = V_{11},$
$T_5 = U_{20}, T_6 = U_{21}, T_7 = V_{20}, T_8 = V_{21},$
$T_9 = Z_{21}, T_{10} = Z_{22}, T_{11} = z_{21}, T_{12} = z_{22}$

Γ_1	$T_{10} = T_9 * T_{10}$ * * *	Γ_2	$T_{13} = T_2 * T_{11}$ * $T_6 = -T_6$ $T_{13} = T_6 + T_{13}$
Γ_3	$T_{12} = T_{10} * T_{11}$ * $T_6 = -T_6$ *	Γ_4	$T_{14} = T_1 * T_{11}$ * $T_{14} = -T_{14}$ $T_{14} = T_5 + T_{14}$
Γ_5	$T_{15} = T_2 * T_{13}$ $T_{15} = T_{14} + T_{15}$ * *	Γ_6	$T_{16} = T_{13} * T_{13}$ * * *
Γ_7	$T_{14} = T_{14} * T_{15}$ * * *	Γ_8	$T_{16} = T_1 * T_{16}$ * * *
Γ_9	$T_{17} = T_3 * T_{12}$ $T_{14} = T_{14} + T_{16}$ $T_7 = -T_7$ $T_{17} = T_7 + T_{17}$	Γ_{10}	$T_{18} = T_4 * T_{12}$ * $T_8 = -T_8$ $T_{18} = T_8 + T_{18}$
Γ_{11}	$T_{12} = T_{15} * T_{17}$ $T_{15} = T_{13} + T_{15}$ $T_{12} = -T_{12}$ *	Γ_{12}	$T_{16} = T_{13} * T_{18}$ $T_{18} = T_{17} + T_{18}$ $T_{16} = -T_{16}$ *
Γ_{13}	$T_{15} = T_{15} * T_{18}$ $T_{15} = T_{12} + T_{15}$ $T_{12} = -T_{12}$ $T_{18} = 1 + T_2$	Γ_{14}	$T_{17} = T_{10} * T_{14}$ * $T_7 = -T_7$ *
Γ_{15}	$T_{18} = T_{16} * T_{18}$ $T_{15} = T_{15} + T_{18}$ * *	Γ_{16}	$T_{10} = T_9 * T_{17}$ * $T_8 = -T_8$ *

HECC: Memory Requirement

- In even characteristic:
  - **HCADD needs 15 registers including 4 for the base divisor and 2 for the curve constants.**
  - **HCDBL needs 13 registers and 4 for the curve constants.**
  - **One register is required for the dummy operations.**
  - **Hence a total of 20 registers are required for the implementation.**
- In odd characteristic
  - **HCADD needs 18 registers including 4 for the base divisor.**
  - **HCDBL needs 16 registers and 2 for the curve constants.**
  - **One register is required for the dummy operations.**
  - **Hence a total of 23 registers are required for the implementation.**
- Note that sequential implementation will require lesser number of registers.
- The registers are only of 80 bit in length now.

Security and Performance

- Security against SPA comes from side-channel atomicity. Any countermeasure against DPA allowing affine arithmetic can be used.

- Performance:
  - In Even Characteristic:
    - for both ADD/DBL 1[i] + 16[m] + 14[a]
  - In odd Characteristic :
    - HCADD : 22[m], HCDBL : 22[m]*

* **Including One Dummy**

Thank You