FUJITSU

A Parallelized Elliptic Curve Multiplication and its Resistance Against Side-channel Attacks*

(* Joint work with Tsuyoshi Takagi)

(株)富士通研究所 セキュアコンピューティング研究部
伊豆 哲也

FUJITSU LABORATORIES LTD.

2002年1月26日(土) お茶の水大学理学部

Contents

FUJITSU

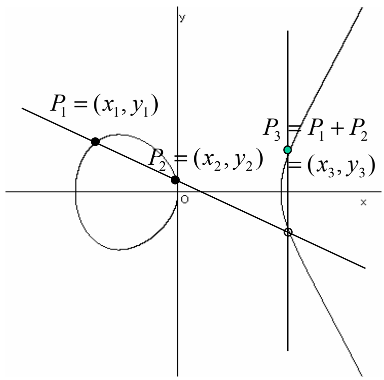
A faster multiplication algorithm resistant against the SCA on both parallel and single computations

- Parallelized computation
  - paralleliziation of an ECADD and an ECDBL
- New addition chain and addition formula
  - x-coordinate-only method
  - (m,m+1)-method
- Resistance against the side-channel attacks (SCA)

Elliptic Curves

FUJITSU

$E / GF(p) : y^2 = x^3 + ax + b$ (Weierstrass form of an elliptic curve)



$$y^2 = x^3 - x$$

Addition (ECADD)

$$x_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)^2 - x_1 - x_2$$

$$y_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)(x_1 - x_3) - y_1$$

Doubling (ECDBL)

$$x_4 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1$$

$$y_4 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_4) - y_1$$

Scalar Multiplication

FUJITSU

To compute $d \times P = \underbrace{P + P + \ldots + P}_{d \text{ times}}$

- Necessary for all elliptic curve-based cryptosystems and the most time-consuming computation.
- How to compute efficiently ?

$100 \times P = \underbrace{P + P + \cdots + P}_{100}$ — 99 ECADDs

$100 \times P = 64 \times P + 32 \times P + 4 \times P$ — 6 ECDBLs + 2 ECADDs

Addition chain: How to combine ECADD/ECDBL

Coordinate system: How to represent ECADD/ECDBL

Addition Chain

FUJITSU

Algorithm 1 (from the MSB)

```
Q[0] = P
for i=n-2 down to 0
  Q[0] = ECDBL(Q[0])
  if d[i]=1 then Q[0] = ECADD(Q[0],P)
return(Q[0])
```

$$d = 2^{n-1} + d[n-2]\times 2^{n-2} + \dots + d[1]\times 2 + d[0]$$

(binary representation of d)

Example

$$d = 100 = 2^6 + 1\times 2^5 + 0\times 2^4 + 0\times 2^3 + 1\times 2^2 + 0\times 2 + 0$$

$P \quad 2\times P \quad 6\times P \quad 12\times P \quad 24\times P \quad 50\times P \quad 100\times P$

$\quad\quad 3\times P \quad\quad\quad\quad\quad\quad 25\times P$

6 ECDBLs + 2 ECADDs

Smart's Idea

FUJITSU

N.Smart (2001)

$E: X^3 + Y^3 + Z^3 = cXYZ$ (Hessian form of an elliptic curve)

- ECADD $(X_3 : Y_3 : Z_3) = (X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2)$

CPU1	CPU2	CPU3
$\lambda_1 = X_2 \times Y_1$	$\lambda_2 = Y_2 \times X_1$	$\lambda_3 = Z_2 \times X_1$
$\lambda_4 = X_2 \times Z_1$	$\lambda_5 = Y_2 \times Z_1$	$\lambda_6 = Z_2 \times Y_1$
$s_1 = \lambda_1 \times \lambda_6$	$s_2 = \lambda_2 \times \lambda_3$	$s_3 = \lambda_5 \times \lambda_4$
$t_1 = \lambda_2 \times \lambda_5$	$t_2 = \lambda_1 \times \lambda_4$	$t_3 = \lambda_6 \times \lambda_3$
$X_3 = s_1 - t_1$	$Y_3 = s_2 - t_2$	$Z_3 = s_3 - t_3$

- Computation Time 12M $\longrightarrow$ 4M

  parallel computation with 3 CPUs

Addition Chain Again

Target: to parallelize ECADD and ECDBL

- But…
  ECADD and ECDBL in Algorithm 1 cannot be parallelized

Algorithm 1 (from the MSB)

```
Q[0] = P
for i=n-2 down to 0
  Q[0] = ECDBL(Q[0])
  if d[i]=1 then Q[0] = ECADD(Q[0],P)
return(Q[0])
```

The value of this Q[0] is determined after ECDBL

Another Chain

Algorithm 2 (from the LSB)

```
Q[0] = P, Q[1] = 0
for i=0 to n-1
  if d[i]=1 then Q[1] = ECADD(Q[0],Q[1])
  Q[0] = ECDBL(Q[0])
return(Q[1])
```

The value of this Q[0] is independent from ECADD

CPU1	CPU2
Q[0] = ECDBL(Q[0])	if d[i]=1 then Q[1] = ECADD(Q[0],Q[1])

- ECADD and ECDBL in Algorithm 2 can be parallelized

Coordinate System

Affine coordinate system ($\mathcal{A}$)

$$E: y^2 = x^3 + ax + b \qquad P = (x, y)$$

ECADD: $x_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)^2 - x_1 - x_2,\ y_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)(x_1 - x_3) - y_1$

ECDBL: $x_4 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1,\ y_4 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_4) - y_1$

- As computing inversions is time-consuming (1I=30～50M), we want to avoid inversions.

Coordinate System (cnt'd)

Projective coordinate system ($\mathcal{P}$) $\quad x = \frac{X}{Z}, y = \frac{Y}{Z}$

$$E: Y^2Z = X^3 + aXZ^2 + bZ^3$$

$$P = (X : Y : Z) \qquad (X : Y : Z) = (rX : rY : rZ)$$

Jacobian coordinate system ($\mathcal{J}$) $\quad x = \frac{X}{Z^2}, y = \frac{Y}{Z^3}$

$$E: Y^2 = X^3 + aXZ^4 + bZ^6$$

$$P = (X : Y : Z) \qquad (X : Y : Z) = (r^2X : r^3Y : rZ)$$

Chudonovsky(-Jacobian) coordinate system ($\mathcal{J}^{\mathcal{C}}$)

$$P = (X : Y : Z : Z^2 : Z^3)$$

the modified Jacobian coordinate system ($\mathcal{J}^m$)

$$P = (X : Y : Z : aZ^4)$$

- No need to compute inversions !

Comparison

	ECADD		ECDBL
	$Z \neq 1$	$Z = 1$	
$\mathcal{A}$	2M+1S+1I	N/A	2M+2S+1I
$\mathcal{P}$	12M+2S	9M+2S	7M+5S
$\mathcal{J}$	12M+4S	8M+3S	4M+6S
$\mathcal{J}^C$	11M+3S	8M+3S	5M+6S
$\mathcal{J}^m$	13M+6S	9M+5S	4M+4S

M: multiplication, S: squaring, I: inversion
in the base field $GF(p)$

Montgomery's Idea

P.Montgomery (1987)

$E : By^2 = x^3 + Ax^2 + x$ (Montgomery form of an elliptic curve)

$P_3 = P_1 + P_2, P_3' = P_1 - P_2, P_4 = 2P_1$

ECADD: $x_3 = \dfrac{(x_1x_2 - 1)^2}{x_3'(x_1 - x_2)^2}$ 3M+2S

ECDBL: $x_4 = \dfrac{(x_1^2 - 1)^2}{4(x_1^3 + Ax_1^2 + x_1)}$ 3M+2S

y-coordinates are not used

x-coordinate-only Method

FUJITSU

x-coordinate-only method for Weierstrass form

- Also discussed by Montgomery.
- No computational advantages were not known.

- Direct translation

$$x_3 \times x_3' = \frac{(x_1x_2 - a)^2 - 4b(x_1 + x_2)}{(x_1 - x_2)^2}, \quad x_4 = \frac{(x_1^2 - a)^2 - 8bx_1}{4(x_1^3 + ax_1 + b)}$$

- Another (additive) formula for ECADD.

$$x_3 + x_3' = \frac{2(x_1 + x_2)(x_1x_2 + a) + 4b}{(x_1 - x_2)^2}$$

Comparison

FUJITSU

	ECADD		ECDBL
	$Z \neq 1$	$Z = 1$	
$\mathcal{A}$	2M+1S+1I	N/A	2M+2S+1I
$\mathcal{P}$	12M+2S	9M+2S	7M+5S
$\mathcal{J}$	12M+4S	8M+3S	4M+6S
$\mathcal{J}^C$	11M+3S	8M+3S	5M+6S
$\mathcal{J}^m$	13M+6S	9M+5S	4M+4S
x (mul)	**9M+2S**	**8M+2S**	**6M+3S**
x (add)	**10M+2S**	**8M+2S**	

But...

FUJITSU

- We need $P_3' = P_1 - P_2$ to compute $P_3 = P_1 + P_2$
  So Algorithm 1/Algorithm 2 cannot be combined.

Algorithm 3 ((m,m+1)-method)

```
Q[0] = P, Q[1] = 2*P
for i=n-2 to 0
  Q[2] = ECDBL(Q[d[i]])
  Q[1] = ECADD(Q[0],Q[1])
  Q[0] = Q[2-d[i]]
  Q[1] = Q[1+d[i]]
return(Q[0])
```

$100 = (1100100)_2$

	P	$2*P$
1	$3*P$	$4*P$
0	$6*P$	$7*P$
0	$12*P$	$13*P$
1	$25*P$	$26*P$
0	$50*P$	$51*P$
0	$100*P$	$101*P$

Another Problem...

FUJITSU

y-recovering is needed

- We have to recover y_d to obtain the whole value of $d \times P = (x_d, y_d)$
- Originally introduced by Agnew-Mullin-Vanstone for the binary field case.

y-recovering for Weierstrass form

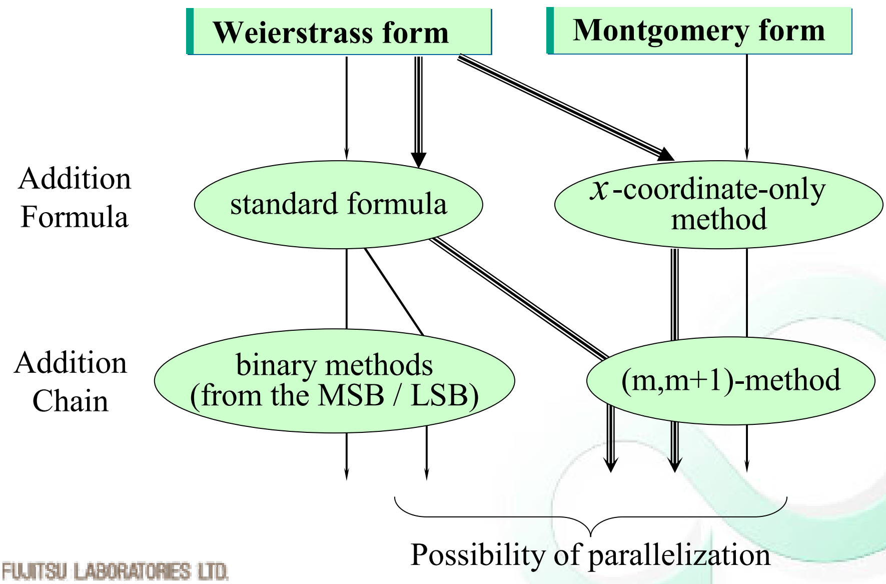
$$y_d = \frac{y^2 + x_d^3 + ax_d + b - (x - x_d)^2 (x + x_d + x_{d+1})}{2y}$$

$100 = (1100100)_2$

	P	$2*P$
1	$3*P$	$4*P$
0	$6*P$	$7*P$
0	$12*P$	$13*P$
1	$25*P$	$26*P$
0	$50*P$	$51*P$
0	$100*P$	$101*P$
	$d \times P$	$(d+1) \times P$

Survey

FUJITSU



Comparison (2CPU)

FUJITSU

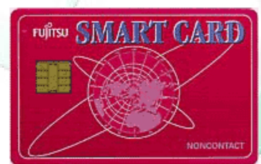
		In each bit	Total (n=160)	
Alg 2	$\mathcal{P}$	12M+2S	1920M+318S+1I	2204.8M
	$\mathcal{J}$	12M+4S	1922M+640S+1I	2464.0M
	$\mathcal{J}^C$	11M+3S	1762M+480S+1I	2176.0M
Alg 3	$\mathcal{P}$	12M+2S	1917M+323S+1I	2205.8M
	$\mathcal{J}$	12M+4S	1916M+643S+1I	2460.8M
	$\mathcal{J}^C$	11M+3S	1758M+484S+1I	2175.2M
	x (mul)	8M+2S	1280M+321S+1I	**1565.8M**
	x (add)	8M+2S	1280M+321S+1I	**1565.8M**

Assumptions: 1S=0.8M, 1I=30M

A non-parallelized method by Lim-Hwang needs 1566.4M

Elliptic Curve Cryptosystem

- Elliptic curve-based cryptosystem achieves higher security with smaller key-length.
- Suitable for implementing in constrained devices such as smart cards and mobile phones.
- The side-channel attacks (SCA) may be applicable if the implementation is naïve or careless.

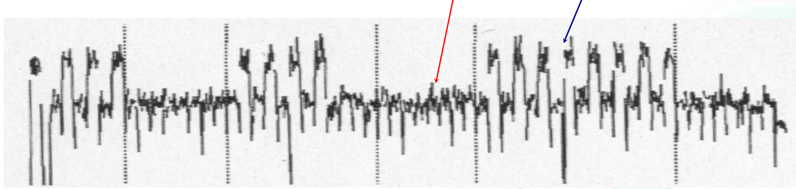


Side-Channel Attacks

Algorithm 1 (from the MSB)

```
Q[0] = P
for i=n-2 down to 0
  Q[0] = ECDBL(Q[0])
  if d[i]=1 then Q[0] = ECADD(Q[0],P)
return(Q[0])
```

The time or the power to execute ECDBL and ECADD are different (side-channel information).



SPA: Simple Power Analysis

DPA: Differential Power Analysis

Countermeasures against SPA

Add-and-double-always method (Coron, 1999)

- Compute ECADD and ECDBL in each bit.

Algorithm 1' (from the MSB)

```
Q[0] = P
for i=n-2 down to 0
  Q[0] = ECDBL(Q[0])
  Q[1] = ECADD(Q[0],P)
  Q[0] = Q[d[i]]
return(Q[0])
```

Algorithm 2' (from the LSB)

```
Q[0] = P, Q[1] = 0
for i=0 to n-1
  Q[2] = ECADD(Q[0],Q[1])
  Q[0] = ECDBL(Q[0])
  Q[1] = Q[1+d[i]]
return(Q[1])
```

- Trade-off: slower complutation

Countermeasures against DPA

Randomized parameters (Coron, 1999)

- Dummy CPU (Coron, 1999)
- Randomized *Z*–coordinate (Coron, 1999)

$$(X : Y : 1) \xrightarrow{randomization} (rX : rY : r)$$

- Randomized curve (Joye-Tymen, 2001)

$$y^2 = x^3 + ax + b \xrightarrow{randomization} y^2 = x^3 + r^4ax + r^6b$$

$$(x : y : 1) \xrightarrow{randomization} (r^2x : r^3y : 1)$$

Security of Algorithm 3

Theorem (Security against the SPA)

Algorithm 3 is as secure as Algorithm 1'/2' against the SPA, if we use a computing archtecture whose swapping power of two variables is negligible.

Theorem (Security against the DPA)

Algorithm 3 with Coron's or Joye-Tymen's countermeasure is as secure as Algorithm 1'/2' against the DPA.

Algorithm 3
((m,m+1)-method)

```
Q[0] = P, Q[1] = 2*P
for i=n-2 to 0
  Q[2] = ECDBL(Q[d[i]])
  Q[1] = ECADD(Q[0],Q[1])
  Q[0] = Q[2-d[i]]
  Q[1] = Q[1+d[i]]
return(Q[0])
```


Comparison (2CPU)

		In each bit	Total (n=160)	
Alg 2' / Coron	$\mathcal{P}$	12M+2S	1924M+320S+1I	2210.0M
	$\mathcal{J}$	12M+4S	1926M+642S+1I	2469.6M
	$\mathcal{J}^C$	11M+3S	1766M+482S+1I	2181.6M
Alg 3 / Coron	x(mul)	9M+2S	1454M+325S+1I	**1744.0M**
	x(add)	10M+2S	1613M+325S+1I	**1903.0M**
Alg 3 / JT	$\mathcal{P}$	12M+2S	1923M+325S+1I	2213.0M
	$\mathcal{J}$	12M+4S	1920M+645S+1I	2466.0M
	$\mathcal{J}^C$	11M+3S	1762M+486S+1I	2180.8M
	x(mul)	8M+2S	1301M+328S+1I	**1593.4M**
	x(add)	8M+2S	1301M+328S+1I	**1593.4M**

Comparison (1CPU)

FUJITSU

		In each bit	Total (n=160)	
Alg 1' / JT	$\mathcal{P}$	16M+7S	2553M+1116S+1I	3475.8M
	$\mathcal{J}$	12M+9S	1917M+1435S+1I	3095.0M
	$\mathcal{J}^{\mathcal{C}}$	13M+9S	2076M+1435S+1I	3254.0M
Alg 2' / JT	$\mathcal{P}$	19M+7S	3049M+1123S+1I	3977.4M
	$\mathcal{J}$	16M+10S	2569M+1604S+1I	3882.2M
	$\mathcal{J}^{\mathcal{C}}$	16M+9S	2569M+1444S+1I	3754.2M
Alg 3 / Coron	x (mul)	15M+5S	2408M+802S+1I	**3079.6M**
	x (add)	16M+5S	2567M+802S+1I	**3238.6M**
Alg 3 / JT	$\mathcal{P}$	19M+7S	3036M+1120S+1I	3962.0M
	$\mathcal{J}$	16M+10S	2556M+1599S+1I	3865.2M
	$\mathcal{J}^{\mathcal{C}}$	16M+9S	2557M+1597S+1I	3739.0M
	x (mul)	14M+5S	2255M+805S+1I	**2929.0M**
	x (add)	14M+5S	2255M+805S+1I	**2929.0M**

(Improved: 2641.8M)

Summary

FUJITSU

A faster multiplication algorithm resistant against the SCA on both parallel and single computations

- Parallelized computation
  - parallelizіation of an ECADD and an ECDBL
- New addition chain and addition formula
  - x-coordinate-only method
  - (m,m+1)-method
- Resistance against the side-channel attacks (SCA)

FUJITSU

THE POSSIBILITIES ARE INFINITE