サイドチャネル耐性を持つ省メモリなスカラー倍算アルゴリズム

木藤 圭亮 †　　宮地 充子 †,‡,§

† 北陸先端科学技術大学院大学 情報科学研究科
923-1292 石川県能美市旭台 1-1
{kitokey, miyaji}@jaist.ac.jp

‡ 大阪大学 大学院工学研究科
565-0871 大阪府吹田市山田丘 2-1

§ 科学技術振興機構 CREST
332-0012 埼玉県川口市本町 4-1-8

あらまし 楕円曲線暗号は従来の暗号方式と比べ短い鍵長で実現することができ，組込み機器などのメモリ量が制限された環境に適する．その主演算であるスカラー倍算は，高速で省メモリかつサイドチャネル攻撃耐性を持つものが望ましい．楕円曲線では複数の座標系が存在するが，一般に逆元が不要な射影座標系を用いる場合が多い．しかし逆元が必要な Affine 座標系に比べ，多くのメモリ量が必要となる．一方で組込み機器ではメモリ量が制限されているため，省メモリ化が重要となる．そこで本研究ではサイドチャネル攻撃耐性を持ち，省メモリかつ高速なスカラー倍算アルゴリズムを提案する．また提案手法を ARM Cortex-M3 へ実装し，評価・比較を行う．
キーワード 楕円曲線暗号，スカラー倍算，サイドチャネル攻撃，組込みシステム，ARM

Memory efficient scalar multiplication algorithms secure against side channel attacks

Keisuke Kito†　　Atsuko Miyaji†,‡,§

†School of Information Science, Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa 923-1292, JAPAN
{kitokey, miyaji}@jaist.ac.jp

‡Graduate School of Engineering, Osaka University
2-1 Yamadaoka, Suita, Osaka 565-0871, JAPAN

§CREST, Japan Science and Technology Agency
4-1-8 Honcho, Kawaguchi, Saitama 332-0012, JAPAN

Abstract Elliptic curve cryptography (ECC) requires smaller key size than traditional cryptosystems, and it is suitable for memory constrained devices such as embedded systems. Scalar multiplication what is dominant computation part of ECC must be fast, memory efficient and secure against side- channel attacks. Typically, projective coordinate is selected because inversion is expensive. But more memory area requires on projective coordinate than affine coordinate. On the other hand, amount of memory in embedded systems are strictly limited. In this research, we propose more memory efficient and faster scalar multiplication algorithm. Furthermore, we implement on ARM Cortex-M3 and then evaluate and compare the paformance of it.
Keywords Elliptic curve cryptography, Scalar multiplication, Side-channel attacks, Embedded systems, ARM

1 はじめに

楕円曲線暗号は従来の暗号方式に比べて，短い鍵長で同等の安全性を実現出来るのが特徴である．そのためメモリ量や計算資源が制限された組込みシステムとの相性がよい．その主演算である有理点のスカラー倍算は，省メモリで高速に演算できることに加えて，サイドチャネル攻撃に対して耐性を持つことが重要である．

また楕円曲線では複数の座標系が存在するが，一般的には逆元演算が不要な射影座標系を用いることが多い．しかし 1 点について 3 つの座標を持つ必要があるため，逆元演算が必要な Affine 座標系に比べて多くのメモリ量が必要とする．

既存のサイドチャネル攻撃耐性を持つスカラー倍算アルゴリズムには，Montgomery Ladder [8, 5] や Joye's Double-add [3]，さらに Joye's m-ary Ladder [4] がある．これらは単純電力解析攻撃，故障利用攻撃に対して耐性を持つ．

また Joye's m-ary Ladder[4] は，スカラー値 k を m 進数に展開してスカラー倍を行うアルゴリズムであり，m を増やすことで高速に演算できるが，使用メモリ量が増大してしまう．

そこで本稿では，Marc Joye によって提案された Joye's m-ary Right-to-Left の $m = 2$ の場合において，2 べき点の演算に Affine 座標系における Double-Quadruple[7] を適用することで，より高速なスカラー倍算アルゴリズムを提案する．また，提案手法を組込みシステムで広く用いられている ARM Cortex-M3 へ実装し，サイクル数とメモリ量の評価もあわせて行う．

2 準備

2.1 節では楕円曲線とその加法について，2.2 節ではスカラー倍算に対するサイドチャネル攻撃とその対策についてそれぞれ説明する．

2.1 楕円曲線

素体 $\mathbb{F}_p (p \geq 5)$ 上の Weierstrass 標準形の楕円曲線は以下の 3 次式で定義される．

$$\begin{aligned} E: \quad & y^2 = x^3 + ax + b \\ & (a, b \in \mathbb{F}_p, 4a^3 + 27b^2 \neq 0) \end{aligned} \tag{1}$$

曲線上の有理点の集合 $E(\mathbb{F}_p)$ と無限遠点 $\mathcal{O}$ を合わせた集合で，点同士の加法が定義できる．

2.1.1 楕円曲線の加法

楕円曲線上の 2 点 $P = (x_1, y_1), Q = (x_2, y_2)$ の加法は以下の加法公式を用いて計算できる．

$$x_3 = \lambda^2 - x_1 - x_2 \tag{2}$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \tag{3}$$

ここで λ は 2 点を結ぶ直線の傾きであり，

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & (P \neq Q) \\ \dfrac{3x_1^2 + a}{2y_1} & (P = Q) \end{cases} \tag{4}$$

と定義される．また $P \neq Q$ の場合は加算公式，$P = Q$ の場合は 2 倍算公式と呼ばれる．

加法公式を繰り返し用いることで，スカラー倍算を計算することができる．スカラー倍算は，楕円曲線上の点 P とスカラー値 k に対して以下のように定義される．

$$kP = \underbrace{P + P + \cdots + P}_{k \text{ 個}} \tag{5}$$

また kP と P から k を求める問題は，楕円曲線上の離散対数問題とよばれ，その求解困難性が楕円曲線暗号の安全性の根拠となっている．

スカラー倍算を求めるアルゴリズムの最も典型的な例として，バイナリ法 (Alg.1) がある．

Alg. 1 Right-to-Left binary method[2]

Input: $P, k = \sum_{i=0}^{n-1} k_i 2^i \ (k_i = \{0, 1\})$
Output: kP
$R[0] \leftarrow \mathcal{O}$, $R[1] \leftarrow P$
for $i = 0$ to $n - 1$ **do**
  if $(k_i = 1)$ **then** $R[0] \leftarrow R[0] + R[1]$
  $R[1] \leftarrow 2R[1]$
return $R[0]$

2.2 スカラー倍算に対するサイドチャネル攻撃

本節ではスカラー倍算アルゴリズムに対するサイドチャネル攻撃について述べる．

2.2.1　単純電力解析 (SPA) 攻撃 [6]

1997 年に Kocher は，演算中の消費電力から秘密鍵などの情報を解析できることを示した．この攻撃は単純電力解析 (Simple Power Analysis, SPA) 攻撃と呼ばれる．先述したバイナリ法 (Alg.1) では，$k_i = 1$ の場合のみ加算が行われるので，消費電力に有意な差が生まれてしまう．

そこで SPA 攻撃を防ぐために提案されたのが，Double-and-Add always である．

Alg. 2 Double-and-Add always[1]

Input: $P, k = \sum_{i=0}^{n-1} k_i 2^i \ (k_i = \{0,1\})$
Output: kP
1: $R[0] \leftarrow \mathcal{O}$, $R[2] \leftarrow P$
2: **for** $i = 0$ to $n-1$ **do**
3: 　$R[1-k_i] \leftarrow R[0] + R[2]$
4: 　$R[2] \leftarrow 2R[2]$
5: **return** $R[0]$

Double-and-Add always はループ内で毎回加算も行うので SPA 攻撃に対して耐性を持つが，後述する Safe-error 攻撃に対しては安全でない．

2.2.2　Safe-error 攻撃 (SEA)[11]

2000 年に Yen らは Double-and-Add always に対する攻撃手法を発見した．この攻撃は Safe-error 攻撃 (SEA) と呼ばれる．Alg.2 では，$k_i = 0$ の場合に 3 行目がダミー演算となってしまう．攻撃者は演算中に，スカラー値に対して誤りを注入する．注入された部分がダミー演算の場合は，出力結果に影響しないため，その部分がダミー演算かどうかが識別できる．

SEA はダミー演算を含まないアルゴリズムを用いることで回避できる．

3　先行研究

本節では 3.1 節に Affine 座標系におけるいくつかの加法公式を紹介する．3.2 節ではサイドチャネル攻撃に耐性を持つスカラー倍算アルゴリズムを紹介する．

3.1　加法公式

本節では，Affine 座標系における 2^n 倍点を求める加法公式を紹介する．

2001 年に坂井らは，Affine 座標系において 2^n 倍算を 1 回の逆元で行う加法公式を発見した．この加法公式では $n \geq 2$ において一般化されており，M, S, I をそれぞれ乗算，2 乗算，逆元演算の計算コストとすると，$(4n+1)M + (4n+1)S + 1I$ の計算量で求めることができる．

Alg. 3 Point quadruple on $\mathcal{A}$[10]

Input: $P = (x_1, y_1)$, Curve parameter a
Output: $R = 4P = (x_3, y_3)$
$A_1 \leftarrow x_1$; $C_1 \leftarrow y_1$
$B_1 \leftarrow 3A_1^2 + a$; $A_2 \leftarrow B_1^2 - 8A_1C_1^2$
$C_2 \leftarrow B_1(4A_1C_1^2 - A_2) - 8C_1^4$
$B_2 \leftarrow 3A_2^2 + 16aC_1^4$; $A_3 \leftarrow B_2^2 - 8A_2C_2^2$
$C_3 \leftarrow B_2(4A_2C_2^2 - A_3) - 8C_2^4$
if $(C_1C_2 = 0)$ **then return** $\mathcal{O}$
$I \leftarrow (4C_1C_2)^{-1}$; $x_3 \leftarrow A_3I^2$; $y_3 \leftarrow C_3I^3$s
return (x_3, y_3)

さらに 2012 年に Le らは，坂井らの加法公式よりも高速に 4 倍算を求める加法公式を発見した．この加法公式では $8M + 8S + 1I$ の計算量で 4 倍点を求めると同時に，2 倍点も求めることができる．本稿ではこの加法公式を Double-Quadruple(DQ) と呼ぶ．

Alg. 4 Double-Quad (DQ) on $\mathcal{A}$[7]

Input: $P = (x_1, y_1)$, Curve parameter a
Output: $2P = (x_2, y_2), 4P = (x_4, y_4)$
$A \leftarrow x_1^2$; $B \leftarrow 3x_1^2 + a$; $C \leftarrow 2y_1^2$; $E \leftarrow C^2$
$F \leftarrow (x_1 + C)^2 - A - E$
$d \leftarrow B(3F - B^2) - 2E$
if $(d = 0)$ **then return** $\mathcal{O}$
$D \leftarrow (2y_1)d$; $I \leftarrow D^{-1}$; $\lambda_1 = dIB$
$x_2 \leftarrow \lambda_1^2 - 2x_1$; $y_2 \leftarrow \lambda_1(x_1 - x_2) - y_1$
$H \leftarrow 3x_2^2 + a$; $\lambda_2 \leftarrow 2EIH$
$x_4 \leftarrow \lambda_2^2 - 2x_2$; $y_4 \leftarrow \lambda_2(x_2 - x_4) - y_2$
return (x_2, y_2, x_4, y_4)

3.2 サイドチャネル攻撃耐性を持つスカラー倍算アルゴリズム

本節では，サイドチャネル攻撃に耐性を持つスカラー倍算アルゴリズムを紹介する．

1987 年に Montgomery はモンゴメリ曲線向けの，高速なスカラー倍算アルゴリズムを開発した．このアルゴリズムは Montgomery Ladder と呼ばれ，当初はスカラー倍算を高速に行う目的で開発された．その後，2002 年に Joye らによって Montgomery Ladder が，ループ内で毎回同じ演算を行い，ダミー演算を持たないため，SPA 攻撃と SEA に対して耐性を持つことが発見された．Montgomery Ladder を Alg.5 に示す．

Alg. 5 Montgomery Ladder[8, 5]

Input: $P, k = \sum_{i=0}^{n-1} k_i 2^i \ (k_i = \{0, 1\})$
Output: $Q = kP \in E(\mathbb{F}_p)$
$R[0] \leftarrow \mathcal{O}$; $R[1] \leftarrow P$
for $i = n-1$ to 0 **do**
  $R[1-k_i] \leftarrow R[1-k_i] + R[k_i]$
  $R[k_i] \leftarrow 2R[k_i]$
return $R[0]$

2007 年に Joye はサイドチャネル攻撃耐性を持つスカラー倍算アルゴリズムを開発した．このアルゴリズムは Joye's Double-Add と呼ばれ，SPA 攻撃と SEA に対して耐性を持つ．Joye's Double-Add のアルゴリズムを Alg.6 に示す．

Alg. 6 Joye's Double-Add[3]

Input: $P, k = \sum_{i=0}^{n-1} k_i 2^i \ (k_i = \{0, 1\})$
Output: $Q = kP \in E(\mathbb{F}_p)$
$R[0] \leftarrow \mathcal{O}$; $R[1] \leftarrow P$
for $i = 0$ to $n-1$ **do**
  $R[1-k_i] \leftarrow 2R[1-k_i] + R[k_i]$
return $R[0]$

2009 年に Joye はスカラー値 k を m 進に展開した上で，スカラー倍算を行うアルゴリズムを開発した．このアルゴリズムは Joye's m-ary Ladder と呼ばれ，SPA 攻撃と SEA に対して耐性を持つ．m はユーザが任意で決めることができかつ，Right-to-left と Left-to-right があるので，実装形態に柔軟性を持たせることができる．

メインのアイディアは m 進数で展開されたスカラー $k = \sum_{i=0}^{n-1} k_i m^i \ (k_i = \{0, \cdots m-1\})$ を

$$k = (k_{n-1} - 1)m^{n-1} + \sum_{i=0}^{n-2} (k_i + m - 1)m^i + 1$$

として考えて行うことである．第一項では，最上位の桁をわざと 1 を減じてある．第二項では最上位以外の各桁を $k_i + m - 1$ とおくことで，その桁に 1 を加えたときに次の桁に繰り上がる状態にする．第三項では最下位に 1 を加えることで最下位で繰り上がりを発生し，最上位まで伝搬させて第一項で減じた 1 を回復している．

R-L，L-R ともにループ回数は $\lceil \log_m k \rceil - 1$ 回，内部変数は $m+1$ 個が必要であり，m を増加させることで，ループ回数が減るので計算量を削減できるが，メモリ量が増加してしまう．

Alg. 7 Joye's m-ary Right-to-Left[4]

Input: $P, k = \sum_{i=0}^{n-1} k_i m^i \ (k_i = \{0, \cdots, m-1\})$
Output: $Q = kP \in E(\mathbb{F}_p)$
for $i = 1$ **to** m **do** $R[i] \leftarrow \mathcal{O}$
$R[0] \leftarrow P$
for $i = 0$ to $n-2$ **do**
  $R[1+k_i] \leftarrow R[1+k_i] + R[0]$
  $R[0] \leftarrow mR[0]$
$R[0] \leftarrow (k_{n-1}-1)R[0] + \sum_{i=1}^{m} (m+i-2)R[i]$
$R[0] \leftarrow R[0] + P$
return $R[0]$

Alg. 8 Joye's m-ary Left-to-Right[4]

Input: $P, k = \sum_{i=0}^{n-1} k_i m^i \ (k_i = \{0, \cdots, m-1\})$
Output: $Q = kP \in E(\mathbb{F}_p)$
for $i = 1$ **to** m **do** $R[i] \leftarrow (m+i-2)P$
$R[0] \leftarrow (k_{n-1} - 1)P$
for $i = n-2$ to 0 **do**
  $R[0] \leftarrow mR[0] + R[1+k_i]$
$R[0] \leftarrow R[0] + P$
return $R[0]$

4　提案手法

本節では Joye's m-ary Ladder の Right-to-Left を，より省メモリかつ高速に行うスカラー倍算アルゴリズムを提案する．

4.1　主なアイディア

本節では提案アルゴリズムの概要を説明する．

Joye's m-ary Ladder は L-R，R-L ともに $m+1$ 個の内部変数が必要だった．今回は Joye's m-ary Right-to-Left の $m = 2$ の場合をベースに，2 倍算について 2-bit を同時に走査することを行う．Alg.9 に提案手法のアルゴリズムを示す．

Alg. 9 2-bit Right-to-Left Ladder

Input: $P, k = \sum_{i=0}^{n-1} k_i 2^i$ ($n \geq 4$, n is even)
Output: $Q = kP$
1: $R[2] \leftarrow k_0P$; $R[3] \leftarrow P$
2: $\{R[1], R[0]\} \leftarrow \mathrm{DQ}(P) = \{4P, 2P\}$
3: **for** $i = 1$ to $(n/2) - 1$ **do**
4: 　$R[2 + k_{2i-1}] \leftarrow R[2 + k_{2i-1}] + R[0]$
5: 　$R[2 + k_{2i}] \leftarrow R[2 + k_{2i}] + R[1]$
6: 　$\{R[1], R[0]\} \leftarrow \mathrm{DQ}(R[1])$
7: $R[0] \leftarrow 2R[3] + R[2]$
8: **return** $R[0]$

ここで 2, 6 行目にある DQ(P) は，Alg.4 で示した，Affine 座標系の Double-Quadruple であり，$\{4P, 2P\} \leftarrow \mathrm{DQ}(P)$ のように表記する．

座標系の取り方として，本稿では以下の 2 つを提案する．

1. すべて Affine 座標系でもつ方法
2. $R[0], R[1]$ を Affine 座標系でもち，$R[2], R[3]$ を Jacobian 座標系でもつ方法

以降，提案 Alg.1，提案 Alg.2 と呼ぶ．

提案 Alg.1 の場合は 4,6 行目，または 5,6 行目において，Montgomery Trick を用いて逆元演算を 1 つに統合することができる．今回は 5, 6 行目について逆元演算の統合を行う．詳細なアルゴリズムを Alg.10 に示す．加算と DQ の逆元を統合したときの計算量は $22M + 1I$，使用メモリ量は 6×28B である．

Alg. 10 2-bit Right-to-Left Ladder (All $\mathcal{A}$)

Input: $P, k = \sum_{i=0}^{n-1} k_i 2^i$ ($n \geq 4$, n is even)
Output: $Q = kP$
1: $R[2] \leftarrow k_0P$; $R[3] \leftarrow P$
2: $\{R[1], R[0]\} \leftarrow \mathrm{DQ}(P) = \{4P, 2P\}$
3: **for** $i = 1$ to $(n/2) - 1$ **do**
4: 　$R[2 + k_{2i-1}] \leftarrow R[2 + k_{2i-1}] + R[0]$
5: 　$\left\{ \begin{array}{l} \text{/* Using Montgomery trick */} \\ \text{/* with } 22M + 1I,\ 6\times 28\text{B */} \\ R[2 + k_{2i}] \leftarrow R[2 + k_{2i}] + R[1] \\ \{R[1], R[0]\} \leftarrow \mathrm{DQ}(R[1]) \end{array} \right.$
6: $R[0] \leftarrow 2R[3] + R[2]$
7: **return** $R[0]$

提案 Alg.2 の場合は 4,5 行目において，Mix-Addition を用いることができる．

4.2　性能検討

本節ではメモリ量と計算量の観点から，提案アルゴリズムの性能検討を行う．

メモリ量の観点では，本アルゴリズムでは 4 個の内部変数が必要であり，$m = 4$ の場合の 5 個より少ないが，$m = 2$ の場合の 3 個より多い．

提案 Alg.1 と 2 を比較すると，提案 Alg.1 は全て Affine 座標系でもつので，有限体 $\mathbb{F}_p$ のビット長が n-bit の場合，点を保持する一時変数だけで $8n$-bit のメモリ量が必要となる．同様に提案 Alg.2 は，$R[0], R[1]$ を Affine 座標系で，$R[2], R[3]$ を Jacobian 座標系でもつので，一時変数だけで $10n$-bit 必要となる．

計算量の観点では，$m = 4$ の場合は 1-bit あたり，0.5 回の加算と 4 倍算が必要である．同様に $m = 2$ の場合は 1 回の加算と 2 倍算，提案手法は 1 回の加算と 0.5 回の DQ がそれぞれ必要となる．$m = 4$ の場合と比較すると 1-bit あたり 0.5 回の加算だけ多いので，4-ary R-L のほうが高速である．$m = 2$ の場合と提案手法は加算回数は同じであるが，2 倍算 1 回に比べて，提案手法は 0.5 回の DQ である．つまり 2 倍算

表 1: 加算回数 (1-bit あたり)・内部変数の比較

Alg.	加算	2^n 倍算	内部変数
提案手法	1 回	DQ 1 回	4 個
2-ary R-L	1 回	Dbl 1 回	3 個
4-ary R-L	1/2 回	Quad 1 回	5 個

2 回分の計算量を，DQ1 回分の計算量が下回れば，提案手法のほうが効率的であると言える．

5 実験

本節では既存手法と提案手法の比較を，任意点のスカラー倍算の演算サイクル数と使用メモリ量で行う．5.1 節では実験に用いた曲線パラメータとプラットフォーム環境について述べる．5.2 節では実験結果について述べる．

5.1 パラメータ・実験環境

本節では実験に用いた曲線パラメータとプラットフォーム環境について述べる．

今回の実験に用いる楕円曲線は表 2 に示す，NIST によって提唱されている 224-bit の曲線 Curve P-224 を用いた．NIST Curve は ECDSA などで広く使用されており，大きな特徴として素体の p に擬メルセンヌ素数用いられているので，乗算後のモジュロを数回の加減算で行うことができる．また曲線パラメータを $a = -3$ とすることで，Jacobian 座標系の加法公式における計算量削減をはかっている．

表 2: 楕円曲線パラメータ (NIST P-224[9])

$p =$	$2^{224} - 2^{64} - 1$
$a =$	-3
$b =$	`b4050a85 0c04b3ab f5413256`
	`5044b0b7 d7bfd8ba 270b3943 2355ffb4`

今回の実験環境には，組込みシステムで広く使用されている ARM Cortex-M3 を用いた．有限体上の演算に用いる多倍長演算は独自のライブラリを用い，今回はプログラム領域を節約するために，有限体上の 2 乗算は使用せずに乗算で代用することとする．すなわち以降の計算量評価は，$S = M$ で行う．使用したプラットフォーム環境は表 3 に示す通りである．

表 3: ARM Cortex-M3 の実験環境

Processor	MB9AF312K
ROM/RAM	(128 + 32)/16 [kB]
IDA	IAR EW for ARM 7.40

5.2 実験結果

本節では先述したプラットフォーム環境において，有限体上の基本演算, 加法公式, スカラー倍算の実験結果について述べる．

はじめに，有限体上の基本演算の演算サイクル数の測定を行う．10^4 回の基本演算における平均サイクル数を測定結果とする．結果を表 4 に示す．なお Inv 以外の基本演算は任意の数値について，一定のサイクル数で演算を行う．実験環境の I/M 比は約 12.87 である．

表 4: $\mathbb{F}_p$ 上の基本演算 平均サイクル数

基本演算	平均 Cycle 数
Add with Carry	134.00
Sub with Borrow	114.00
Mul with Mod	2,040.00
Inv	26,199.38
I/M 比	12.87

次に加法公式の性能測定を行う．10^3 回の加法を行い，その平均演算サイクル数を結果とする．使用メモリ量は加法に必要な一時メモリ量を指す．演算サイクル数と使用メモリの結果を表 5, 6, 7 に示す．今回の実験環境の場合，Affine 座標系加算と Jacobian 座標系加算の演算サイクル数がほぼ同程度である．また Affine 座標系 2 倍算の演算サイクル数は，Jacobian 座標系 2 倍算の 2 倍程度となっている．また Affine 座標系の Double-Add と Jacobian 座標系 Double-Add の演算サイクル数もほぼ同程度である．

表 5: 加算公式 平均サイクル数・メモリ量

加法公式	計算量	Cycle 数	メモリ量
$\mathcal{A}+\mathcal{A}$	$3M+I$	32,947.25	3×28B
$\mathcal{J}+\mathcal{A}$	$11M$	22,984.00	4×28B
$\mathcal{J}+\mathcal{J}$	$16M$	33,110.00	4×28B

表 6: 2^n 倍算公式 平均サイクル数・メモリ量

加法公式	計算量	Cycle 数	メモリ量
$2\mathcal{A}$	$4M+I$	35,283.43	3×28B
DQ($\mathcal{A}$)	$16M+I$	61,229.37	5×28B
$2\mathcal{J}$	$8M$	17,948.00	4×28B

スカラー倍算の実験の前に，今回の実験における座標系のもち方を表 8 に示す．また，m-ary R-L($\mathcal{A},\mathcal{J}$) と m-ary L-R($\mathcal{A},\mathcal{J}$) は，それぞれ Alg.7 中の 4 行目，Alg.8 中の 4 行目を Mix-Addition で行う．同様に提案 Alg.2 も，Alg.9 中の 4, 5 行目を Mix-Addition で行うこととする．

最後に任意点のスカラー倍算の演算サイクル数の測定を行う．今回は 10^3 回の任意点のスカラー倍算における平均サイクル数を測定結果とする．使用メモリ量は，加法の時に必要な一時メモリ量，スカラー倍算の時に必要な点を保持するための一時メモリ量，スカラー値 k を保持するメモリ量の総計を示している．今回は提案手法のほか，比較のために Montgomery Ladder, Joye's m-ary R-L, L-R($m=2,4$) についても測定を行った．スカラー倍算の演算サイクル数と総使用メモリ量の測定結果を表 9 に示す．

はじめに 2 つの提案手法の比較を行う．1 ループあたりの計算量で比較すると，提案 Alg.1 は $25M+2I$ であり，提案 Alg.2 は $38M+1I$ である．提案 Alg.1 のほうが効率的になるためには $I/M<13$ である必要がある．実験環境の I/M 比は約 12.87 であるので，提案 Alg.1 のほうが効率的となる．実測結果も提案 Alg.1 のほうが高速かつ省メモリである．

次に m-ary R-L の比較を行う．$m=2$ の場合，1 ループあたりの計算量は ($\mathcal{A},\mathcal{J}$) の場合 $15M+1I$，($\mathcal{J}$) の場合 $24M$ であり，$I/M=9$ を境界に計算量が反転する．$m=4$ の場合，1 ループあたりの計算量は ($\mathcal{A},\mathcal{J}$) の場合 $27M+1I$，

表 7: Double-Add 平均サイクル数・メモリ量

加法公式	計算量	Cycle 数	メモリ量
$2\mathcal{A}+\mathcal{A}$	$11M+I$	49,925.83	5×28B
$2\mathcal{J}+\mathcal{A}$	$18M$	37,838.00	8×28B
$2\mathcal{J}+\mathcal{J}$	$21M$	49,034.00	8×28B

表 8: 各アルゴリズムにおける座標系のもち方

Algorithm	Affine	Jacobian
Mont.Lad. ($\mathcal{J}$)	–	$R[0],R[1]$
m-ary L-R($\mathcal{A}$)	$R[0]$ – $R[m]$	–
m-ary L-R($\mathcal{A},\mathcal{J}$)	$R[1]$ – $R[m]$	$R[0]$
m-ary R-L($\mathcal{A},\mathcal{J}$)	$R[0]$	$R[1]$ – $R[m]$
m-ary R-L($\mathcal{J}$)	–	$R[0]$ – $R[m]$
提案 Alg. 1.	$R[0]$ – $R[3]$	–
提案 Alg. 2.	$R[0],R[1]$	$R[2],R[3]$

($\mathcal{J}$) の場合 $32M$ であり，$I/M=5$ を境界に計算量が反転する．今回の I/M 比では，($\mathcal{J}$) の場合のほうが高速となる．演算サイクル数も ($\mathcal{J}$) のほうが高速であり，メモリ量は ($\mathcal{A},\mathcal{J}$) に比べて 28B 増加で抑えられている．

次に m-ary L-R の比較を行う．$m=2$ の場合，1 ループあたりの計算量は ($\mathcal{A},\mathcal{J}$) の場合 $11M+1I$，($\mathcal{J}$) の場合 $21M$ であり，$I/M=10$ を境界に計算量が反転する．$m=4$ の場合，1 ループあたりの計算量は ($\mathcal{A},\mathcal{J}$) の場合 $27M+1I$，($\mathcal{J}$) の場合 $32M$ であり，$I/M=7$ を境界に計算量が反転する．今回の I/M 比では，Jacobian 座標系のみを用いたときのほうが高速となる．演算サイクル数も ($\mathcal{J}$) のほうが高速であるが，メモリ量は ($\mathcal{A},\mathcal{J}$) に比べて 4×28B 増加している．

次に提案 Alg.1 と m-ary R-L の比較を行う．演算サイクル数は 4-ary R-L($\mathcal{J}$) が最も高速であり，次いで 4-ary R-L($\mathcal{A},\mathcal{J}$)，2-ary R-L($\mathcal{J}$) であり，その次に提案 Alg.1 が高速である．4.2 節で解説したように，加法回数の観点から 4-ary R-L よりも高速にはならないが，$I/M<11.5$ ならば 2-ary R-L($\mathcal{J}$) よりも提案 Alg.1 が高速となる．今回の I/M 比は 11.5 を下回らないので，演算サイクル数も 2-ary R-L($\mathcal{J}$) を下回らなかった．メモリ量では，提案 Alg.1 が最も省メモリである．

表 9: スカラー倍算 平均サイクル数・メモリ量

Algorithm	Cycles	Mem.
Mont.Lad. ($\mathcal{J}$)	11,433,628.94	11×28B
2-ary L-R($\mathcal{A}$)	11,151,457.06	12×28B
提案 Alg. 1.	11,986,427.97	15×28B
2-ary L-R($\mathcal{A}, \mathcal{J}$)	8,499,699.39	16×28B
4-ary L-R($\mathcal{A}$)	9,668,105.60	16×28B
2-ary R-L($\mathcal{A}, \mathcal{J}$)	13,002,911.04	17×28B
2-ary R-L($\mathcal{J}$)	11,395,289.90	18×28B
提案 Alg. 2.	12,026,117.07	19×28B
4-ary L-R($\mathcal{A}, \mathcal{J}$)	6,413,756.54	20×28B
4-ary R-L($\mathcal{A}, \mathcal{J}$)	9,671,971.67	26×28B
4-ary R-L($\mathcal{J}$)	7,944,724.12	27×28B

6 おわりに

本稿では，Joye's m-ary Right-to-Left の $m = 2$ の場合において，2 べき点の演算に Affine 座標系における Double-Quadruple を適用し，さらに Montgomery Trick により逆元回数を削減することで，より高速なスカラー倍算アルゴリズムを提案した．また ARM Cortex-M3 で実装し，Joye's m-ary R-L の $m = 2$ よりも高速で，$m = 4$ よりも省メモリなアルゴリズムであることを実証した．

今後の課題として，提案手法の改良のほかに Jacobian 座標系の加法部分を Co-Z を用いた場合にどのような影響を与えるかを調査したい．

謝辞

本研究は JSPS 科研費 15K00183, 15K00189 の助成を受けたものです。

参考文献

[1] Jean-Sebastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *CHES 1999*, LNCS 1717, pp. 292–302. 1999.

[2] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004.

[3] M. Joye. Highly regular right-to-left algorithms for scalar multiplication. In *CHES 2007*, LNCS 4727, pp. 135–147. 2007.

[4] M. Joye. Highly regular m-ary powering ladders. In *SAC 2009*, LNCS 5867, pp. 350–363. 2009.

[5] Marc Joye and Sung-Ming Yen. The montgomery powering ladder. In *CHES 2002*, LNCS 2523, pp. 291–302. 2003.

[6] PaulC. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO 96*, LNCS 1109, pp. 104–113. 1996.

[7] Duc-Phong Le and Binh P. Nguyen. Fast point quadrupling on elliptic curves. SoICT '12, pp. 218–222, New York, NY, USA, 2012. ACM.

[8] Peter L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, Vol. 48, No. 177, pp. 243–264, 1987.

[9] NIST. Recommended elliptic curves for federal government use. `http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf`, 1999.

[10] Y. SAKAI and K. SAKURAI. Efficient scalar multiplications on elliptic curves with direct computations of several doublings : Special section on cryptography and information security. *IEICE trans.*, Vol. 84, No. 1, pp. 120–129, jan 2001.

[11] Sung-Ming Yen and M. Joye. Checking before output may not be enough against fault-based cryptanalysis. *Computers, IEEE Transactions on*, Vol. 49, No. 9, pp. 967–970, Sep 2000.