SCIS 2009 The 2009 Symposium on Cryptography and Information Security
Otsu, Japan, Jan. 20-23, 2009
The Institute of Electronics, Information and Communication Engineers

Skew-Frobenius 写像を利用した種数 3 の超楕円曲線上の整数倍算
Scalar multiplications on genus 3 hyperelliptic curves by using skew-Frobenius maps

小崎 俊二 * 松尾 和人 *

Abstract— 本論文では、高速な暗号系が構成されることが期待される skew-Frobenius 写像を利用した種数 3 の超楕円曲線上の整数倍算について、multi-exponentiation に対する事前計算量の削減を行なうことによる整数倍算の高速化を検討する。Multi-exponentiation の計算方法として知られている simultaneous 法と interleave 法それぞれに対して skew-Frobenius 写像の性質を利用して事前計算量の削減を行なうことで整数倍算の高速化を図る。更に 32bitCPU に対して skew-Frobenius 写像を利用した 160 ビット鍵長の整数倍算の実装実験を行ない、通常の NAF_w を用いた整数倍算と比較して約 13%実行速度が高速化することを確認した。

Keywords: hyperelliptic curve cryptosystems, scalar multiplications, skew-Frobenius expansions, multi-exponentiations

1 はじめに

超楕円曲線暗号 [10] は同等の安全性をもつ楕円曲線暗号と比較して、曲線の定義体のサイズを小さくとることが可能である。種数 3 の超楕円曲線暗号に関しては、64 ビット CPU に適したサイズの素体上定義された曲線を利用した高速な実装が知られている [13, 6, 27]。

超楕円曲線暗号の高速化においては、整数倍算の高速化が重要である。高速な整数倍算の計算方法として、部分体曲線を用いる方法 [3, Section 15.1] が知られている。この方法は曲線の定義体 $\mathbb{F}_p$ の拡大体 $\mathbb{F}_{p^n}$ 上定義された有理点群に対する p 乗 Frobenius 写像による整数展開を利用して整数倍算を高速化するものである。しかし、この方法を利用する場合、$\mathbb{F}_{p^n}$ 有理点群は非自明な部分群 ($\mathbb{F}_p$ 有理点群) を持つため、素数位数の有理点群を利用する場合と比較して $\mathbb{F}_{p^n}$ のサイズが大きくなる。特に、拡大次数 n が小さい場合には $\mathbb{F}_{p^n}$ 有理点群に対して非自明な部分群のサイズが大きくなるため、$\mathbb{F}_{p^n}$ の演算コストが増加する。

Skew-Frobenius 写像 [8, 11, 12] は、$\mathbb{F}_p$ 上定義された曲線の $\mathbb{F}_{p^n}$ 上の 2 次ツイストに対して定義される自己同型写像である。Skew-Frobenius 写像は、Frobenius 写像と同様の性質をもち、更に素数位数の有理点群上で利用可能である。従って、Frobenius 写像を利用した場合と比較して曲線の定義体を小さくできるため、skew-Frobenius 写像を利用することによる整数倍算の高速化が期待される。

Skew-Frobenius 写像を利用した整数倍算は、skew-Frobenius 展開の各項を独立した元と考えることにより multi-exponentiation[3, Section 9.1.5] を適用することで、群の 2 倍算の回数を削減可能である。Multi-exponentiation の計算方法には、simultaneous 法 [26, 4, 24] と interleave 法 [15, 21, 9] が知られている。それぞれの方法に対して群の逆元演算が容易である場合について、事前計算を利用することで群の加算回数を削減する整数表現が提案されている。Simultaneous 法に対する整数表現としては、joint sparse form (JSF) [23] とそれを一般化した colexicographically minimal integer representations (CMR_w) [7] が知られている。Interleave 法に対する整数表現としては、non-adjacent form (NAF) [16, 19] と表現に利用する整数の範囲を拡張した width-w non-adjacent form (NAF_w) [14, 22] が知られている。これらの方法は、逆元演算が容易な一般の群の multi-exponentiation に関する方法である。一方 Frobenius 写像を利用した整数倍算において、写像の性質を利用した事前計算の計算量削減法が提案されている [1]。従って、skew-Frobenius 展開について multi-exponentiation を適用する場合にも、skew-Frobenius 写像の性質を利用することで整数倍算を効率的に行なうことが可能であると考えられる。

本論文では、skew-Frobenius 写像を利用した種数 3 の超楕円曲線上の整数倍算に multi-exponentiation を適用した場合の事前計算量の削減を行なう。Simultaneous 法と interleave 法それぞれに対して、skew-Frobenius 展開の性質を利用した事前計算量の削減を行ない、それぞれの方法の群演算量が最小となるようにすることで高速な整数倍算を構成する。更に、この整数倍算の群演算量評価を利用し 32bit CPU に対する 160 ビット鍵長の整数倍算のパラメータ選択を行ない、実装実験により skew-Frobenius 写像を利用した整数倍算の効果を確認する。

本論文の構成を以下に示す。2 節では、skew-Frobenius 写像と multi-exponentiation を用いた整数倍算についてまとめる。3 節では、interleave 法と simultaneous 法それぞれに対する事前計算量の削減を行なう。更に、4 節

* Institute of Information Security, 2-14-1, Tsuruya-cho Kanagawa-ku, Yokohama 221-0835, Japan

では、32 ビット CPU に対する種数 3 の超楕円曲線の実装に利用するパラメータの選択を行ない、その整数倍算の実験結果を示す。最後に 5 節では、skew-Frobenius 写像を利用した種数 3 の超楕円曲線上の整数倍算についてまとめる。

2 準備

本節では、種数 3 の超楕円曲線上の skew-Frobenius 写像の定義と性質についてまとめる。また整数倍算に利用される multi-exponentiation の計算アルゴリズムの計算方法をまとめる。

Skew-Frobenius 写像は Frobenius 写像と比較して高々 6 回の乗算の増加で計算可能であり、種数 3 の超楕円曲線に対しその群演算と比較して効率的に計算可能である。更に、曲線の定義体として高速な実装が可能な OEF[2] の利用を考えた場合に、skew-Frobenius 写像は Frobenius 写像と比較してコンパクトなサイズの定義体を利用可能であることから、skew-Frobenius 写像を利用した整数倍算は高速なソフトウェア実装が期待される。この skew-Frobenius 写像を利用した整数倍算は multi-exponentiation として計算可能である。multi-exponentiation には大きく 2 つの方法が知られており、更に事前計算を利用することで効率的な整数倍算を行なう方法が知られている。

2.1 種数 3 の超楕円曲線上の skew-Frobenius 写像

p を奇素数として、有限体 $\mathbb{F}_p$ 上定義された種数 3 の超楕円曲線 C を、

$$C : Y^2 = F(X),\ F(X) \in \mathbb{F}_p[X] \tag{1}$$

と定義する。ここで、F は重根をもたない monic な 7 次多項式とする。自然数 n に対して、C の $\mathbb{F}_{p^n}$ 上の 2 次ツイストを C_t とする。更に、C と C_t それぞれの Jacobian を $\mathbb{J}_C$ と $\mathbb{J}_{C_t}$ と書き、それぞれの $\mathbb{F}_{p^n}$ 有理点群を $\mathbb{J}_C(\mathbb{F}_{p^n})$ と $\mathbb{J}_{C_t}(\mathbb{F}_{p^n})$ と書く。

$\mathbb{J}_C$ 上の p 乗 Frobenius 写像を ϕ_p とするとき、skew-Frobenius 写像 $\tilde{\phi}_p$ は、

$$\tilde{\phi}_p : \mathbb{J}_{C_t} \xrightarrow{\cong} \mathbb{J}_C \xrightarrow{\phi_p} \mathbb{J}_C \xrightarrow{\cong} \mathbb{J}_{C_t}$$

で定義される $\mathbb{J}_{C_t}$ 上の自己同型写像である [11, 12]。$\tilde{\phi}_p$ は ϕ_p と比較して高々6 回の $\mathbb{F}_{p^n}$ 乗算の増加のみで計算可能であり、群演算と比較して効率的に計算可能な写像である。また、$\tilde{\phi}_p$ は $\mathbb{J}_{C_t}(\mathbb{F}_{p^n})$ 上の非自明な自己同型写像であり ϕ_p と同一の特性多項式を満足することから、自己同型環 $\mathrm{End}(\mathbb{J}_C(\mathbb{F}_{p^n}))$ の部分環 $\mathbb{Z}[\phi_q]$ において ϕ_p と同様な整数展開を行なうことが可能である。更に、$n = 2^i, i \in \mathbb{N}$ であるとき $\mathbb{J}_{C_t}(\mathbb{F}_{p^n})$ の位数が素数である C_t が存在する。従って、Frobenius 写像 ϕ_p を利用した場合に鍵長を $\sharp(\mathbb{J}_C(\mathbb{F}_{p^n})/\mathbb{J}_C(\mathbb{F}_p)) \approx p^{3(n-1)}$ とする必要があることと比較して、$\tilde{\phi}_p$ を利用した場合には $\sharp\mathbb{J}_{C_t}(\mathbb{F}_{p^n}) \approx p^{3n}$ を鍵長とすることが可能である。特に $\mathbb{F}_{p^n}$ として OEF[2] のように n が小さい体を利用する場合には $\mathbb{F}_{p^n}$ のサイズの増加を抑えることができるため、skew-Frobenius 写像の利用は整数倍算を高速化するものと期待される。

2.2 Skew-Frobenius 展開と multi-exponentiation

任意の $\mathcal{D} \in \mathbb{J}_{C_t}(\mathbb{F}_{p^n})$ に対して $\tilde{\phi}_p$ は、

$$\tilde{\phi}_p^n(\mathcal{D}) = -\mathcal{D} \tag{2}$$

であることから、$\tilde{\phi}_p$ を利用した整数展開はその長さを n 以下とすることが可能である。即ち $k \in \mathbb{Z}$ とするとき整数倍算 $k\mathcal{D}$ の skew-Frobenius 展開は、

$$k\mathcal{D} = \sum_{i=0}^{n-1} k_i \tilde{\phi}_p^i(\mathcal{D}),\ k_i \in \mathbb{Z} \tag{3}$$

と書ける。式 (3) の右辺は、$\mathcal{D}, \tilde{\phi}_p(\mathcal{D}), \cdots, \tilde{\phi}_p^{n-1}(\mathcal{D}) \in \mathbb{J}_{C_t}(\mathbb{F}_{p^n})$ を独立した元と考えると multi-exponentiation [3, Section 9.1.5] を用いることで計算可能である。式 (3) の各 k_i は p^3 程度の大きさであることから、単一の k に対して binary 法 [3, Section 9.1.1] などの一般の exponentiation を適用した場合と比較して、skew-Frobenius 展開に multi-exponentiation を用いると整数倍算における $\mathbb{J}_{C_t}(\mathbb{F}_{p^n})$ の 2 倍算の回数を削減可能である。

Multi-exponentiation の計算法として、

- Simultaneous 法 [26, 4, 24]
- Interleave 法 [15, 21, 9]

が知られている。Simultaneous 法のアルゴリズムの概要を Algorithm 1 に示す。

Algorithm 1 Simultaneous 法

Input: $k_0, \ldots, k_{n-1} \in \mathbb{Z}$, $\mathcal{D}_0, \ldots, \mathcal{D}_{n-1} \in \mathbb{J}_{C_t}(\mathbb{F}_{p^n})$
Output: $\mathcal{D} = \sum_{i=0}^{n-1} k_i \mathcal{D}_i \in \mathbb{J}_{C_t}(\mathbb{F}_{p^n})$
1: ℓ を $k_0, \ldots, k_{n-1}$ の中の最大ビット長とし、
   $k_{i,j}$ を $k_i = \sum_{j=0}^{\ell-1} k_{i,j} 2^j$ と 2 進表現する
   /*事前計算*/
2: **for all** $(d_0, \ldots, d_{n-1}) \in \{0,1\}^n \setminus \{(0, \ldots, 0)\}$ **do**
3: $\quad \mathcal{D}_{(d_0,\ldots,d_{n-1})} \leftarrow \sum_{i=0}^{n} d_i \mathcal{D}_i$
   /*主計算*/
4: $\mathcal{D} \leftarrow \mathcal{D}_{(k_{0,\ell-1},\ldots,k_{n-1,\ell-1})}$
5: **for** $j = \ell - 2$ down to 0 **do**
6: $\quad \mathcal{D} \leftarrow 2\mathcal{D}$
7: $\quad$ **if** $(k_{0,j}, \ldots, k_{n-1,j}) \neq (0, \ldots, 0)$ **then**
8: $\quad\quad \mathcal{D} \leftarrow \mathcal{D} + \mathcal{D}_{(k_{0,j},\ldots,k_{n-1,j})}$
9: **return** $\mathcal{D}$

Simultaneous 法は、Algorithm 1 のステップ 2,3 において事前計算を行い、主計算部分のステップ 8 においてこの事前計算結果を用いて multi-exponentiation を計算する。即ち、$k_0, \ldots, k_{n-1}$ の 2 進表現の同一桁について 1 回の 2 倍算と $k_0, \ldots, k_{n-1}$ の同一桁がすべて 0 でなければ加算を 1 回行なうことを繰り返して multi-exponentiation を計算する。一方、interleave 法のアルゴリズムの概要を Algorithm 2 に示す。

Algorithm 2 Interleave 法

Input: $k_0, \ldots, k_{n-1} \in \mathbb{Z}$, $\mathcal{D}_0, \ldots, \mathcal{D}_{n-1} \in \mathbb{J}_{C_t}(\mathbb{F}_{p^n})$
Output: $\mathcal{D} = \sum_{i=0}^{n-1} k_i \mathcal{D}_i \in \mathbb{J}_{C_t}(\mathbb{F}_{p^n})$
1: ℓ を $k_0, \ldots, k_{n-1}$ の中の最大ビット長とし、
$k_{i,j}$ を $k_i = \sum_{j=0}^{\ell-1} k_{i,j} 2^j$ と 2 進表現する
2: $\mathcal{D} \leftarrow \mathcal{D}_m$, m は $k_{m,\ell-1} \neq 0$ を満足する最小の数
3: **for** $i = m+1$ to $n-1$ **do**
4: **if** $k_{i,\ell-1} \neq 0$ **then**
5: $\mathcal{D} \leftarrow \mathcal{D} + \mathcal{D}_i$
6: **for** $j = \ell - 2$ down to 0 **do**
7: $\mathcal{D} \leftarrow 2\mathcal{D}$
8: **for** $i = 0$ to $n-1$ **do**
9: **if** $k_{i,j} \neq 0$ **then**
10: $\mathcal{D} \leftarrow \mathcal{D} + \mathcal{D}_i$
11: **return** $\mathcal{D}$

Interleave 法は、$k_0, \ldots, k_{n-1}$ の 2 進表現の同一桁について 1 回の 2 倍算と同一桁中で 0 でないものをそれぞれ加算することを繰り返して multi-exponentiation を計算する。

それぞれの方法に対して、群の逆元演算が容易である場合に k_0,…,k_{n-1} に対して符号付 2 進表現を利用することにより加算の群演算回数を削減可能である。更に、k_0,…,k_{n-1} の表現に利用する整数の範囲を拡張して事前計算を利用することにより、加算の群演算回数を削減可能である。

Simultaneous 法においては、$k_0, \ldots, k_{n-1}$ の同一桁がすべて 0 となるものが多ければ Algorithm 1 のステップ 8 における加算が削減できる。$n = 2$ 即ち、k_0, k_1 に対して同一桁がすべて 0 でない桁の数 (joint Hamming weight) が最小となる符号付 2 進表現として joint sparse form (JSF) [23] が知られている。更に JSF を $n > 1$ に一般化した表現として colexicographically minimal integer representation (CMR_w) [7] が知られている。CMR_w は、同時に k_0,…,k_{n-1} の桁表現に利用する整数の範囲を拡張することで、joint Hamming weight を削減する。桁表現に利用する整数の集合を $\{d_i \mid -2^{w-1}+1 \leq d_i \leq 2^{w-1}-1\}$ として、その範囲をウィンドウ幅 w を用いて表す。CMR_w を用いた simultaneous 法のアルゴリズムを Algorithm 3 に示す。

Algorithm 3 CMR_w を用いた simultaneous 法

Input: $k_0, \ldots, k_{n-1} \in \mathbb{Z}$, $\mathcal{D}_0, \ldots, \mathcal{D}_{n-1} \in \mathbb{J}_{C_t}(\mathbb{F}_{p^n})$
Output: $\mathcal{D} = \sum_{i=0}^{n-1} k_i \mathcal{D}_i \in \mathbb{J}_{C_t}(\mathbb{F}_{p^n})$
1: $(k_0, \ldots, k_{n-1})$ に対する CMR_w を求め、
$k_i = \sum_{j=0}^{\ell-1} k_{i,j} 2^j$ とする
/*事前計算*/
2: **for all** $\mathbf{d} = (d_0, \ldots, d_{n-1}) \in S$ **do**
3: $\mathcal{D}_{\mathbf{d}} \leftarrow \sum_{i=0}^{n} d_i \mathcal{D}_i$
/*主計算*/
4: $\mathcal{D} \leftarrow \mathrm{sign}(\mathbf{k}_{\ell-1}) \mathcal{D}_{\mathbf{k}_{\ell-1}}$
5: **for** $j = \ell - 2$ down to 0 **do**
6: $\mathcal{D} \leftarrow 2\mathcal{D}$
7: **if** $(k_{0,j}, \ldots, k_{n-1,j}) \neq (0, \ldots, 0)$ **then**
8: $\mathcal{D} \leftarrow \mathcal{D} + \mathrm{sign}(\mathbf{k}_j) \mathcal{D}_{\mathbf{k}_j}$
9: **return** $\mathcal{D}$

ここで Algorithm 3 のステップ 2 の S は、

$$S := \{(x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n \mid 0 \leq i \leq n-1 \text{ に対して} \\ -2^{(w-1)} < x_i < 2^{(w-1)}, \\ \text{少なくとも 1 つの } x_i \text{ は } x_i \equiv 1 \pmod 2, \\ x_i \neq 0 \text{ を満足する最小の } i \text{ に対して } x_i > 0\} \quad (4)$$

で与えられる集合とする。更にステップ 4 と 8 では、j を固定したとき $k_{i,j} \neq 0$ を満足する最小の i に対して $k_{i,j}$ の符号を s_j として、$\mathbf{k_j} := (s_j k_{0,j}, \ldots, s_j k_{n-1,j}) \in S$ と書き、$\mathrm{sign}(\mathbf{k_j}) := s_j$ とする。

一方、interleave 法は simultaneous 法とは違い $k_0, \ldots, k_{n-1}$ それぞれについて 0 の桁数が多い整数表現を用いれば、Algorithm 2 のステップ 10 における加算が削減できる。0 でない桁数 (Hamming weight) が最小である符号付 2 進表現として NAF [16, 19] が知られている。更に、桁表現に利用する整数の範囲を拡張して Hamming weight を削減する整数表現として width-w non-adjacent form (NAF_w) [14, 22] が知られている。NAF_w を用いた interleave 法のアルゴリズムを Algorithm 4 に示す。

Algorithm 4 NAF_w を用いた Interleave 法

Input: $k_0, \ldots, k_{n-1} \in \mathbb{Z}$, $\mathcal{D}_0, \ldots, \mathcal{D}_{n-1} \in \mathbb{J}_{C_t}(\mathbb{F}_{p^n})$
Output: $\mathcal{D} = \sum_{i=0}^{n-1} k_i \mathcal{D}_i \in \mathbb{J}_{C_t}(\mathbb{F}_{p^n})$
1: $k_0, \ldots, k_{n-1}$ それぞれに対して NAF_w を求め、
$k_i = \sum_{j=0}^{\ell-1} k_{i,j} 2^j$ とする
/*事前計算*/
2: **for all** $d \in \{x \mid 0 < x < 2^{w-1},\ x \equiv 1 \bmod 2\}$ **do**
3: **for** $i = 0$ to $n-1$ **do**
4: $\mathcal{D}_{i,d} \leftarrow d\mathcal{D}_i$
/*主計算*/
5: $\mathcal{D} \leftarrow \mathrm{sign}(k_{m,\ell-1}) \mathcal{D}_{m,|k_{m,\ell-1}|}$, m は $k_{m,\ell-1} \neq 0$ を満足する最小の数
6: **for** $i = m+1$ to $n-1$ **do**
7: **if** $k_{i,\ell-1} \neq 0$ **then**
8: $\mathcal{D} \leftarrow \mathcal{D} + \mathrm{sign}(k_{i,\ell-1}) \mathcal{D}_{i,|k_{i,\ell-1}|}$
9: **for** $j = \ell - 2$ down to 0 **do**
10: $\mathcal{D} \leftarrow 2\mathcal{D}$
11: **for** $i = 0$ to $n-1$ **do**
12: **if** $k_{i,j} \neq 0$ **then**
13: $\mathcal{D} \leftarrow \mathcal{D} + \mathrm{sign}(k_{i,j}) \mathcal{D}_{i,|k_{i,j}|}$
14: **return** $\mathcal{D}$

3 Skew-Frobenius 写像を利用した整数倍算

本節では、整数表現として CMR_w を用いた simultaneous 法と NAF_w を用いた interleave 法それぞれに対する事前計算量の削減を行なう。

どちらの方法を skew-Frobenius 写像を利用した整数倍算に適用した場合にも、利用する整数表現の範囲を決めるパラメータであるウィンドウ幅 w を選択する必要がある。w を大きくとると事前計算部分の群演算回数は増加するが主計算部分の群演算回数は減少するため、整数倍算全体の群演算回数が少なくなるように、k のサイズに応じて w を選択する必要がある。従って、事前計算量の削減は、メモリ使用量を削減するだけでなく、同一のメモリ使用量のもとで多くの事前計算を行なえることに

より総群演算回数を削減する効果がある。即ち事前計算部分の群演算量の削減は整数倍算の高速化にとって重要な要素となる。

以降では、skew-Frobenius 写像を利用した整数倍算に対する simultaneous 法と interleave 法それぞれについて、skew-Frobenius 展開の性質を利用することで事前計算量の削減を行う。

3.1 Simultaneous 法の事前計算量削減

CMR_w を用いた simultaneous 法に対して事前計算が必要な元の全体は、(4) 式の集合 S を用いて

$$T = \left\{ \sum_{i=0}^{n-1} d_i \tilde{\phi}_p^i(\mathcal{D}) \mid (d_0, \ldots, d_{n-1}) \in S \right\}$$

となる。従って、通常の multi-exponentiation では事前計算が必要な元の個数は $\mathcal{D}, \tilde{\phi}_p(\mathcal{D}), \ldots, \tilde{\phi}_p^{n-1}(\mathcal{D})$ も含めて $\frac{1}{2}\left((2^w-1)^n - (2^{w-1}-1)^n\right)$ である。事前計算は、$0 \le i \le n-1$ それぞれに対して $\tilde{\phi}_p^i(\mathcal{D})$ に順次 $\tilde{\phi}_p^i(\mathcal{D})$ を加算することで、$2\tilde{\phi}_p^i(\mathcal{D}), \cdots, (2^{w-1}-1)\tilde{\phi}_p^i(\mathcal{D})$ を計算し、次に残りの $d_0\mathcal{D} + \cdots + d_{n-1}\tilde{\phi}_p^{n-1}\mathcal{D} \in T$ を計算することで実行可能である。この事前計算には $(2^{w-1} - 2^{w-2} - 2)n + \frac{1}{2}((2^w-1)^n - (2^{w-1}-1)^n)$ 回の群加算が必要である。

式 (2) より、

$$\begin{aligned}
\tilde{\phi}_p\left(\sum_{i=0}^{n-1} d_i\tilde{\phi}_p^i(\mathcal{D})\right) &= -d_{n-1}\mathcal{D} + d_0\tilde{\phi}_p(\mathcal{D}) \\
&\quad + \cdots + d_{n-2}\tilde{\phi}_p^{n-1}(\mathcal{D}) \\
\tilde{\phi}_p^2\left(\sum_{i=0}^{n-1} d_i\tilde{\phi}_p^i(\mathcal{D})\right) &= -d_{n-2}\mathcal{D} - d_{n-1}\tilde{\phi}_p(\mathcal{D}) \\
&\quad + \cdots + d_{n-3}\tilde{\phi}_p^{n-1}(\mathcal{D}) \\
&\vdots \\
\tilde{\phi}_p^n\left(\sum_{i=0}^{n-1} d_i\tilde{\phi}_p^i(\mathcal{D})\right) &= \sum_{i=0}^{n-1} -d_i\tilde{\phi}_p^i(\mathcal{D}) \\
&\vdots \\
\tilde{\phi}_p^{2n}\left(\sum_{i=0}^{n-1} d_i\tilde{\phi}_p^i(\mathcal{D})\right) &= \sum_{i=0}^{n-1} d_i\tilde{\phi}_p^i(\mathcal{D})
\end{aligned}$$

であることを利用すると、事前計算が必要な元の個数を $1/n$ とすることが可能である。即ち skew-Frobenius 展開に対して事前計算が必要な元の個数は $\mathcal{D}$ も含めて $\frac{1}{2n}((2^w-1)^n - (2^{w-1}-1)^n)$ と削減される。この事前計算は $(2^{w-2}-2) + \frac{1}{2n}((2^w-1)^n - (2^{w-1}-1)^n)$ 回の群加算で行なうことが可能である。

3.2 Interleave 法の事前計算量削減

NAF_w を用いた interleave 法に対して事前計算が必要な元の全体は、Algorithm 4 のステップ 2-4 より

$$\{d\tilde{\phi}_p^i(\mathcal{D}) \mid 0 \le i \le n-1,\ 0 < d < 2^{(w-1)},\ d \equiv 1 \bmod 2\}$$

である。

$\tilde{\phi}_p$ の性質を利用すると、次のように事前計算量を削減可能である。$\tilde{\phi}_p$ が効率的に計算可能であることから、主計算部分で必要となる $i \ge 1$ に対する $d\tilde{\phi}_p^i(\mathcal{D})$ は $\tilde{\phi}_p^i(d\mathcal{D})$ としてその都度主計算部分で $d\mathcal{D}$ から計算すればよいので、事前計算が必要な元の全体は、

$$\{d\mathcal{D} \mid 1 \le d \le 2^{(w-1)} - 1,\ d \equiv 1 \bmod 2\}$$

と削減可能である。これらの元は、$\mathcal{D}$ から $2\mathcal{D}$ を計算した後、$\mathcal{D}$ に対して順次 $2\mathcal{D}$ を加算することで計算可能である。従って、事前計算は 1 回の 2 倍算と $2^{w-2}-1$ 回の加算で計算可能である。

4 種数 3 の超楕円曲線上の整数倍算の実装

本節では、携帯機器などに広く利用されている 32 ビット CPU に対して skew-Frobenius 写像を利用した整数倍算をソフトウェア実装し、skew-Frobenius 写像を利用した整数倍算の効果を確認する。

最初に、32 ビット CPU に対して高速なソフトウェア実装が可能な曲線のパラメータ選択を行なう。更に、前節で得られた結果を用いて種数 3 の超楕円曲線上の整数倍算の群演算量を評価し、整数倍算が効率的に行なえるウィンドウ幅を選択する。そして、選択したパラメータを用いて skew-Frobenius 写像を利用した整数倍算を実装し、実行時間を計測することで効果を確認する。

4.1 曲線パラメータの選択

鍵長を 160 ビットとして、32 ビット CPU に対する曲線パラメータの選択をする。

曲線の定義体 $\mathbb{F}_{p^n}$ は高速なソフトウェア実装が期待される OEF [2] を利用することとした。ここで、p は高速な素体演算が可能であることからメルセンヌ素数とし、32 ビット以内で最大の $p = 2^{31}-1$ を選択した。種数 3 の超楕円曲線暗号に対する攻撃法として Pollard の rho 法 [18] の計算量 $O(p^{\frac{3n}{2}})$ より少ない $O(p^{\frac{4n}{3}})$ の計算量もつ攻撃法 [17, 5] が知られている。この攻撃法を考慮すると p^n は 60 ビット程度とする必要がある。従って拡大次数は $n = 2$ を選択した。特に、高速な拡大体上の乗算が可能であることから、$\mathbb{F}_{p^n}$ の多項式表現は $\mathbb{F}_{p^n} \cong \mathbb{F}_p[X]/(X^2+1)$ を選択した。

曲線 C_t のパラメータは、skew-Frobenius 写像の特長を利用して $\sharp\mathbb{J}_{C_t}(\mathbb{F}_{p^n})$ が素数であるものを選択した。即ち式 (1) の曲線 C をランダムに与え $\mathbb{J}_C(\mathbb{F}_p)$ の Frobenius 写像 ϕ_p の特性多項式 $\chi(X) \in \mathbb{Z}[X]$ を計算し、$\sharp\mathbb{J}_{C_t}(\mathbb{F}_{p^n})$ が素数となるものを探索することで、C_t のパラメータを選択した。

4.2 ウィンドウ幅 w の選択

$n = 2$ とし、式 (3) の展開係数 k_0 と k_1 の最大ビット長を ℓ としたときの整数倍算の群加算コストを Table 1 に示す。

Table 1: $n = 2$ の場合における simultaneous 法と interleave 法それぞれの事前計算・主計算部分の群加算回数

	事前計算部分	主計算部分
Simul.	$(2^{w-2}-2) + \frac{3\cdot 2^{2(w-1)} - 2^w}{4}$	$\frac{(3\cdot 2^w + 4)(\ell-1)}{3w2^w + 2^w + 4w - 4}$
Inter.	$2^{w-2} - 1$	$\frac{2\ell - 1}{w+1}$

Table 1 の各行において、Simul. は simultaneous 法を、Inter. は interleave 法を表す。また、Table 1 の各列は事前計算部分と主計算部分それぞれにおける群加算回数を示している。Simultaneous 法と interleave 法それぞれの主計算部分における群 2 倍算の回数を $\ell-1$ とし、群加算と 2 倍算のコストが同一であるとしてそれぞれの方法の総群演算回数を評価した。$\ell \approx \lceil \log_2 p^3 \rceil = 93$ としてそれぞれの方法に対する総群演算回数が最小となる w を求めると、simultaneous 法は $w=3$、interleave 法は $w=5$ であった。このときの総群演算回数はそれぞれ、simultaneous 法が 131.3 回、interleave 法が 130.8 回であった。

4.3 実装結果

4.2 節で総群演算回数が少なかった interleave 法について 4.1 節で選択したパラメータを用いて実装を行なった。Table 2 に実装に利用した曲線パラメータを示す。

Table 2: 実装に利用した曲線パラメータ

$p = 2^{31} - 1$

$\mathbb{F}_{p^n} \cong \mathbb{F}_p[X]/(X^2+1)$

$$\begin{aligned} C_t : Y^2 \quad = \quad & X^7 + (1102784164 + 2099349210\alpha)X^5 \\ & +(147978801 + 1115837825\alpha)X^4 \\ & +(1922630011 + 1004370797\alpha)X^3 \\ & +(1088566058 + 1890525800\alpha)X^2 \\ & +(1064534894 + 1994659678\alpha)X \\ & +(1352297621 + 545614834\alpha) \end{aligned}$$

$$\begin{aligned} C : Y^2 \quad = \quad & X^7 + 863946643X^5 + 1372526599X^4 \\ & +231832837X^3 + 1668274122X^2 \\ & +2024621019X + 600616318 \end{aligned}$$

$\tilde{\phi}_p$ の特性多項式:

$$\begin{aligned} X^6 \quad & +57995X^5 + 2217371019X^4 + 79726867182563X^3 \\ & +4761768002634226293X^2 \\ & +267454730389609733218955X \\ & +9903520300447984150353281023 \end{aligned}$$

$$\sharp \mathbb{J}_{C_t}(\mathbb{F}_{p^n}) = 980797143186008309259073799764189323630026862407102652 73$$

有限素体演算は [6] に記述された方法に従って実装した。拡大体 $\mathbb{F}_{p^n}$ の乗算は、本実装環境において Karatsuba 乗算が効率的ではなかったため、次のように classical 乗算を実装した。

Input: $a = a_0 + a_1 X, b = b_0 + b_1 X \in \mathbb{F}_p[X]/(X^2+1)$
Output: $c = c_0 + c_1 X$ s.t. $c = ab$
1: $c_0 = a_0 b_0 - a_1 b_1$
2: $c_1 = a_0 b_1 + a_1 b_0$

$\mathbb{F}_{p^n}$ の逆元演算は、次のように実装した。

Input: $a = a_0 + a_1 X \in \mathbb{F}_p[X]/(X^2+1),\ a \neq 0$
Output: $c = c_0 + c_1 X$ s.t. $c = 1/a$
1: $N = a_0^2 + a_1^2 \in \mathbb{F}_p$
2: $c_0 = a_0 / N$
3: $c_1 = -a_1 / N$

また、種数 3 の超楕円曲線上の加算・2 倍算のアルゴリズムは [27] の Toom 乗算を用いたアルゴリズムを利用した。更に skew-Frobenius 展開において必要な多倍長整数演算には NTL[20] を利用した。

実装環境は Table 3 に示したものを使用した。実装において直接アセンブラ言語を使用せず言語拡張の関数のみを利用した。また、SIMD 命令は利用しなかった。

Table 3: 実装環境

CPU	Intel XScale PXA270 520MHz (ARMv5TE)
OS	Windows Mobile 6 Classic (CE OS 5.2.1433)
Compiler	Microsoft C++ Compiler ver. 15.00.20720

整数倍算の実行時間の測定は、ランダムに選んだベースポイント $\mathcal{D}$ とランダムに選んだ 160 ビットの整数 1000 組それぞれに対して整数倍算を 100 回繰り返し 1 回当りの平均時間を求めることで行なった。

実行時間の測定結果を Table 4 を示す。比較対象として、同一の環境のもとで skew-Frobenius 展開を利用せず単一の k に対して NAF_w を利用した整数倍算の実行時間も Table 4 に示す。

Table 4: 160 ビット整数に対する整数倍算の実行時間比較

Skew-Frobenius 展開 (NAF_w) を利用した場合	2.83 ms
展開を利用せず NAF_w のみを利用した場合	3.26 ms

Table 4 より、skew-Frobenius 写像を利用した整数倍算は skew-Frobenius 写像を利用しない場合と比較して約 13%実行時間が短縮されることが判る。しかし、理論値では約 32%程度の群演算回数の削減が見込めることから今回の実装は理論値と比較して劣っている。これは skew-Frobenius 展開において多倍長整数の汎用ライブラリを利用していることが原因であると考えられ、今後改良が必要であると考えられる。

次に、種数 3 の超楕円曲線の整数倍算の実装として知られているもの中で、同様の CPU アーキテクチャに対するもの [25] との比較を Table 5 に示す。

Table 5: ARM アーキテクチャに対する整数倍算の実装比較

	$\mathbb{F}_{p^n}$	Mcycle	ms	CPU	Clock
[25]	$\mathbb{F}_{2^{63}}$	8.48	106ms	ARM7TDMI	80MHz
本論文	$\mathbb{F}_{(2^{31}-1)^2}$	1.47	2.83ms	XScale	520MHz

Table 5 より本実装は [25] と比較して整数倍算の速度が 6 倍になっていることが判る。

5 まとめ

Skew-Frobenius 写像を利用した種数 3 の超楕円曲線上の整数倍算について、multi-exponentiation の simultaneous 法と interleave 法それぞれを用いた場合の事前計算部分の群演算量を skew-Frobenius 写像の性質を利用して削減することで、整数倍算を効率化した。また、160 ビットの鍵長に対してこの事前計算量の削減を用いた整数倍算全体の群演算量を評価し、simultaneous 法と interleave 法それぞれに対する整数倍算の群演算量を最小にするウィンドウ幅を求めた。更に、32 ビット CPU

に対する曲線パラメータを選択して skew-Frobenius 写像を利用した整数倍算の実装を行ない、通常の NAF_w を利用した場合と比較して整数倍算の実行時間が約 13%短縮することを確認した。

References

[1] K. Aoki, F. Hoshino, and T. Kobayashi. A cyclic window algorithm for ECC defined over extension fields. In S. Qing, T. Okamoto, and J. Zhou, editors, *Information and Communications Security*, number 2229 in Lecture Notes in Computer Science, pages 62–73. Springer-Verlag, 2001.

[2] D. V. Bailey and C. Paar. Optimal extension fields for fast arithmetic in public-key algorithms. In H. Krawczyk, editor, *Advances in Cryptology - CRYPTO '98*, number 1462 in Lecture Notes in Computer Science, pages 472–485. Springer-Verlag, 1998.

[3] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography.* Chapman & Hall/CRC, 2005.

[4] T. ElGamal. A public key cryptosystem and sigunature scheme based on discrete logarithms. *IEEE Trans. on Info. Theory*, IT-31(4):469–472, 1985.

[5] P. Gaudry, E. Thomé, N. Thériault, and C. Diem. A double large prime variation for small genus hyperelliptic index calculus. *Math. Comp.*, 76(257):475–492, 2007.

[6] M. Gonda, K. Matsuo, K. Aoki, J. Chao, and S. Tsujii. Improvements of addition algorithm on genus 3 hyperelliptic curves and their implementation. *IEICE Trans.*, E88-A(1), January 2005. 89-96.

[7] C. Heuberger and J.A. Muir. Minimal weight and colexicographically minimal integer representations. *Journal of Mathematical Cryptology*, 1(4):297–328, 2007.

[8] T. Iijima, K. Matsuo, J. Chao, and S. Tsujii. Cnstruction of Frobenius maps of twist elliptic curves and its application to elliptic scalar multiplication. In *Proc. of SCIS2002*, pages 699–702, January 2002.

[9] T. Kobayashi, H. Morita, K. Kobayashi, and F. Hoshino. Fast elliptic curve algorithm combining Frobenius map and table reference to adapt to higher characteristic. In *Advances in Cryptology - EUROCRYPT '99*, number 1592 in Lecture Notes in Computer Science, pages 176–189. Springer-Verlag, 1999.

[10] N. Koblitz. Hyperelliptic curve cryptosystems. *J. Cryptology*, 1(3):139–150, 1989.

[11] S. Kozaki, K. Matsuo, and Y. Shimbara. Skew-Frobenius maps on hyperelliptic curves. In *Proc. of SCIS2007*, number 1D2-4, January 2007.

[12] S. Kozaki, K. Matsuo, and Y. Shimbara. Skew-Frobenius maps on hyperelliptic curves. *IEICE Japan Trans. Fundamentals*, E91-A(7):1839–1843, 2008.

[13] J. Kuroki, M. Gonda, K. Matsuo, J. Chao, and S. Tsujii. Fast genus three hyperelliptic curve cryptosystems. In *Proc. of SCIS2002*, pages 503–507, January 2002.

[14] A. Miyaji, T. Ono, and H. Cohen. Efficient elliptic curve exponentiation. In Y. Han, T. Okamoto, and S. Qing, editors, *Information and Communications Security*, volume 1334 of *Lecture Notes in Computer Science*, pages 282–290. Springer Berlin / Heidelberg, 1997.

[15] B. Möller. Algorithms for multi-exponentiation. In *Selected Areas in Cryptography SAC 2001*, pages 165–180. Springer, 2001.

[16] F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Info. Theory Appl.*, 24:531–543, 1990.

[17] K. Nagao. Index calculus attack for Jacobian of hyperelliptic curves of small genus using two large primes. *Japan Journal of Industrial and Applied Mathematics*, 24(3), 2007.

[18] J. M. Pollard. Monte Carlo methods for index computation (mod p). *Math. Comp.*, 32:918–924, 1978.

[19] G. Reitwiesner. Binary arithmetic. *Adv. in Comp.*, 1:231–308, 1960.

[20] V. Shoup. A tour of NTL. `http://www.shoup.net/ntl/`, 2003.

[21] S.G. Sim and P.J. Lee. An efficient implementation of two-term exponentiation. Technical Report ISEC99-82, IEICE Japan, 2000.

[22] J. A. Solinas. An improved algorihtm for arithmetic on a family of elliptic curves. In *Advances in Cryptology - CRYPTO '97*, number 1294 in Lecture Notes in Computer Science, pages 357–371. Springer-Verlag, 1997.

[23] J.A. Solinas. Low-weight binary representations for pairs of integers. Tech. Rep. CORR 2001-41, University of Waterloo, 2001.

[24] E.G. Straus. Addition chains of vectors. *American Mathematical Monthly*, 71:806–808, 1964.

[25] T. Wollinger, J. Pelzl, and C. Paar. Cantor versus harley: Optimization and analysis of explicit formulae for hyperelliptic curve cryptosystems. *IEEE Trans. Computers*, 54(7):861–872, 2005.

[26] S.-M. Yen, C.-S. Laih, and A.K. Lenstra. Multi-exponentiation. In *Computers and Digital Techniques, IEE Proceedings*, volume 141, pages 325–326, 1994.

[27] 入海 淳, 松尾 和人, 趙 晋輝, and 辻井 重男. 超楕円曲線上の Harley アルゴリズムにおける resultant 計算について. Technical Report ISEC2006-5, 電子情報通信学会, July 2006.