情報処理学会論文誌 Vol. 50 No. 11 2717–2726 (Nov. 2009)

最小の論理命令数での GF(3) 上の加算による η_T ペアリングの高速実装

川原祐人[†1] 青木和麻呂[†2] 高木剛[†1]

近年，ID ベース暗号などのプロトコルが構築可能であることからペアリング暗号が注目され，それにともない，プロトコルの構築に用いるペアリング暗号を効率的に計算することが必要となっている．標数 3 の η_T ペアリングは素体 GF(3) = $\{0, 1, 2\}$ 上の演算を基に構成される．Harrison らは GF(3) の元の 2 ビットを用いた 2 進数への自然なビット割当てと論理命令 OR，XOR を用いて GF(3) 上の加算を 7 命令で構成したが，7 命令が最小の論理命令数であるかどうかは証明されていない．本稿では，GF(3) 上の加算を論理命令数で評価し，最小の論理命令数で命令列を構成するために命令列の全数探索を行った．探索実験より，任意の論理命令やビット割当てを用いても 5 命令以下で GF(3) 上の加算を構成できず，また Harrison らとは異なるビット割当てを用いて 6 命令で GF(3) 上の加算を構成した．さらに Harrison らとは異なるビット割当てによる 6 命令の GF(3) 上の加算を用いて η_T ペアリングの高速実装を行った．128-bit セキュリティと見積もられている GF(3^{509}) 上の η_T ペアリングの計算時間は AMD Opteron（2.2 GHz）上で 16.3 msec であり，従来の 7 命令での GF(3) 上の加算を用いた場合と比較し約 7%高速となった．

Faster Implementation of η_T Pairing Using Minimum Number of Logical Instructions for GF(3)-addition

Yuto Kawahara,[†1] Kazumaro Aoki[†2] and Tsuyoshi Takagi[†1]

Recently, pairing-based cryptosystems have attracted much attention in cryptography, since pairing-based cryptosystems can provide many novel applications such as ID-based cryptosystems. Then pairing-based cryptosystems are required to efficiently compute. The η_T pairing in characteristic three is implemented by arithmetic in GF(3) = $\{0, 1, 2\}$. Harrison, et al. reported an efficient implementation of the GF(3)-addition by using seven logical instructions (consisting of OR and XOR) with the two-bit encoding. It has not yet been proven whether seven is the minimum number of logical instructions for the GF(3)-addition. In this paper, we search the instruction sequences exhaustively for constructing the GF(3)-addition with the minimum number of logical instructions. In our experiment, we construct many implementations of the GF(3)-addition using only six logical instructions with different encodings. We then prove that there is no implementation of the GF(3)-addition using five logical instructions with any encoding of GF(3) by two bits. Moreover, we apply the new GF(3)-additions to an efficient software implementation of the η_T pairing. The running time of the η_T pairing over GF(3^{509}), that is considered to be realized as 128-bit security, using the new GF(3)-addition with the encoding which is different from Harrison, et al. is 16.3 milliseconds on an AMD Opteron (2.2-GHz) processor. This is approximately 7% faster than the implementation using the previous GF(3)-addition with seven logical instructions.

1. はじめに

有限体上の楕円曲線を用いた双線形ペアリングは ID ベース暗号[7),23)] などの新たな暗号プロトコルを構築可能である．Tate ペアリングを効率的に計算可能なアルゴリズムは Miller[19)] により初めて提案された．また Duursma ら[9)] により，標数 2 または 3 の有限体 GF(2^m)，GF(3^m) 上の超特異楕円曲線を用いた効率的な計算アルゴリズムが提案された．Barreto ら[3)] は Duursma-Lee アルゴリズムを改良し，ループ回数を削減することで計算時間が約 2 倍高速となる η_T ペアリングを提案した．現在，GF(3^m) 上の η_T ペアリングは最も高速なペアリングの 1 つである．

GF(3^m) 上の演算を構成するには GF(3) 上の加算が必要となるが，GF(3) 上の加算は x86 アーキテクチャ[16)] などを基とする一般的な CPU において 1 命令で直接的に計算することができない．GF(3) 上の加算の効率的な実装方法として，GF(3) の元を 2 ビットで表現し，論理命令を用いて構成する方法がある．Galbraith ら[10)] は論理命令 AND，OR，XOR，NOT を用いて 12 命令での GF(3) 上の加算の構成を示し，また Harrison ら[15)] は論理命令 OR，XOR を用いて 7 命令での GF(3) 上の加算の構成を示した．現在，7 命令が論理命令による GF(3) 上の加算の構成において最小の論理命令数である．

論理命令を用いた命令列の探索実験として，Osvik[20)] による 4-bit 入出力の S-box を計

†1 公立はこだて未来大学大学院システム情報科学研究科
Graduate School of Systems Information Science, Future University-Hakodate
†2 NTT 情報流通プラットフォーム研究所
NTT Information Sharing Platform Laboratories

© 2009 Information Processing Society of Japan

算する命令列の探索実験がある．彼は x86 アーキテクチャ上での S-box の高速実装を目的としており，x86 アーキテクチャでの実装に特化した命令列の探索を行った．したがって，探索では使用可能な命令として論理命令 AND，OR，XOR，NOT および MOV を扱い，また 5 個のレジスタで計算可能な命令列の探索を行っている．

本稿では，論理命令を用いた構成において，GF(3) 上の加算を 7 命令以下で計算可能な命令列の探索を行う．GF(3) 上の加算命令列の探索では，GF(3) の各元を $\mathrm{GF}(2)^2$ の 2 ビットに割り当てることで GF(3) 上の加算を写像 $\mathrm{GF}(2)^2 \times \mathrm{GF}(2)^2 \to \mathrm{GF}(2)^2$ として扱い，命令列の全数探索を行う．探索結果より，$\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2\}$ などのビット割当て，または ANDN などの追加の論理命令を用いたとき，6 命令で GF(3) 上の加算命令列を構成した．また任意の論理命令と 2 ビットによるビット割当てを用いても，GF(3) 上の加算命令列を 5 命令以下で構成できなかった．したがって，本稿の探索実験において 6 命令が GF(3) 上の加算命令列を構成する最小の論理命令数であった．

さらに本稿では，新たに構成した 6 命令での GF(3) 上の加算を用いて $\mathrm{GF}(3^m)$ 上の演算および η_T ペアリングの実装，評価を行った．文献 2) において，NIST は 2011 年には 80-bit セキュリティより大きな鍵サイズを用いることを推奨しており，本稿では特に 128-bit セキュリティを持つと見積もられている $\mathrm{GF}(3^{509})$ 上の η_T ペアリングを用いて評価を行った[1),18)]．AMD Opteron model 275（2.2 GHz）上において，ビット割当て $\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2\}$ を用いることにより，ビット割当て $\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}$ を用いた場合と比較し $\mathrm{GF}(3^{509})$ 上の加算が約 11%高速となり，同様に，Window 幅 $w = 4$ の Comb 法による $\mathrm{GF}(3^{509})$ 上の乗算が約 8%高速となった．結果として，6 命令での GF(3) 上の加算を用いた $\mathrm{GF}(3^{509})$ 上の η_T ペアリングの計算時間は約 16.3 msec であり，従来の 7 命令での GF(3) 上の加算を用いた場合に比べ約 7%高速となった．

本稿は以下のとおりに構成される．2 章では，GF(3) 上の加算および従来結果を示す．3 章では，GF(3) 上の加算命令列の探索方法を述べる．4 章で $\mathrm{GF}(3^m)$ 上の演算および η_T ペアリングの実装詳細と評価結果を示す．最後に，5 章で本稿をまとめる．

2. GF(3) 上の加算

素体 $\mathrm{GF}(3) = \{0,1,2\}$ は 3 個の元により構成されるため，GF(3) の元 $e \in \mathrm{GF}(3)$ は 2 ビット e_h，$e_l \in \mathrm{GF}(2)$ を用いて，

$$e = (e_h, e_l)$$

と表現することができる．このとき $\mathrm{GF}(2)^2$ から GF(3) の各元への自然な割当てを

$$\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\} \tag{1}$$

とする．

GF(3) の元 a，b が与えられたとき，GF(3) 上の加算 $c \leftarrow a + b$ は $c = a + b \bmod 3$ により計算される．また式 (1) の自然なビット割当てを用いたとき，GF(3) の元 $e = (e_h, e_l)$ に対する負の元 $-e$ は次のように求められる．

$$\begin{cases} (-e)_h & \leftarrow & e_l \\ (-e)_l & \leftarrow & e_h \end{cases}$$

GF(3) 上の減算 $c \leftarrow a - b$ は，$a + (-b)$ と変換することにより加算と同様の計算コストで計算可能である．

本稿では，次の論理命令を用いて GF(3) 上の加算を構成する．

| ：ビットごとの OR 演算
& ：ビットごとの AND 演算
$\wedge$ ：ビットごとの XOR 演算
$\bar{x}$ ：ビットごとの反転 (NOT)

論理命令を用いた効率的な GF(3) 上の加算命令列の構成は非自明な問題である．効率的に計算を行うためには，GF(3) 上の加算命令列を最小の論理命令数により構成することが望ましい．Galbraith ら[10)] は次のような 12 命令での GF(3) 上の加算命令列を示した．

$$\begin{cases} c_h & \leftarrow & ((a_h \wedge b_h) \,\&\, \overline{(a_l \mid b_l)}) \mid (a_l \,\&\, b_l) \\ c_l & \leftarrow & ((a_l \wedge b_l) \,\&\, \overline{(a_h \mid b_h)}) \mid (a_h \,\&\, b_h) \end{cases}$$

さらに Harrison ら[15)] は一時変数 t を用いて，次のような 7 命令での GF(3) 上の加算命令列を示した．

$$\begin{cases} t & \leftarrow & (a_h \mid b_l) \wedge (b_h \mid a_l) \\ c_h & \leftarrow & t \wedge (a_l \mid b_l) \\ c_l & \leftarrow & t \wedge (a_h \mid b_h) \end{cases} \tag{2}$$

これまで数多くの実装で GF(3) 上の加算が用いられているが，ほとんどの実装において Harrison らの命令列が用いられている[1),4),12),22)]．しかし，7 命令が GF(3) 上の加算命令列を構成するために必要な最小の論理命令数であるかどうかは証明されていない．

© 2009 Information Processing Society of Japan

3. GF(3) 上の加算命令列の探索

本章では，最小の論理命令数で GF(3) 上の加算を計算するため，GF(3) 上の加算が計算可能な命令列の探索方法および探索結果を示す．

3.1 GF(3) のビット割当ての選択

2 ビットによるビット表現 $\mathrm{GF}(2)^2$ から素体 GF(3) の元への割当てを Assignment R としたとき，Assignment R は次のように定義される．

$$R = \{(e_h, e_l) \mapsto e \mid e_h, e_l \in \mathrm{GF}(2), e \in \mathrm{GF}(3)\}$$

元 a，$b \in \mathrm{GF}(3)$ に対し，GF(3) 上の加算 $c \leftarrow a+b$ は写像 $\mathrm{GF}(3) \leftarrow \mathrm{GF}(3) \times \mathrm{GF}(3)$ である．この写像の各集合に対する Assignment を明示するため，加算 $c \leftarrow a+b$ の入力 a に対する Assignment を R_a，入力 b に対する Assignment を R_b，出力 c に対する Assignment を R_c とする．また R_a，R_b および R_c の組を Assignment set とする．このとき自然な割当ては $R_a = R_b = R_c = \{(0,0) \mapsto 0,\ (0,1) \mapsto 1,\ (1,0) \mapsto 2\}$ となる[15]．Assignment set の取るパターンは数多く存在する．次に Assignment set の一例を示す．

$$\begin{cases} R_a &= \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\} \\ R_b &= \{(1,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2, (0,0) \mapsto 2\} \\ R_c &= \{(1,1) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\} \end{cases} \tag{3}$$

本稿では，Assignment set の選択にあたり次のような場合を考慮する．

- 冗長表現．GF(3) のある元 e_0 は 2 個の異なるビット表現 (e_{0h}, e_{0l})，$(e'_{0h}, e'_{0l}) \in \mathrm{GF}(2)^2$ を持つことができる．また GF(3) の残りの元 e_1，e_2 はそれぞれ (e_{1h}, e_{1l})，$(e_{2h}, e_{2l}) \in \mathrm{GF}(2)^2$ に割り当てられる．ここで (e_{0h}, e_{0l})，(e'_{0h}, e'_{0l})，(e_{1h}, e_{1l})，(e_{2h}, e_{2l}) はそれぞれ $\mathrm{GF}(2)^2$ において異なる．
- 任意の Assignment の選択．Assignment set R_a，R_b，R_c の各 Assignment は同様である必要はない．

i，$j \in \{a, b, c\}$ に対して，$\#R_i$ を Assignment R_i の要素数，また $R_i \subseteq R_j$ を R_j と R_i の包含関係とする．このとき Assignment set の取るすべてのパターンを次のように分類する．

- $\#R_a = \#R_b = \#R_c = 3$
  - 1–i　自然な Assignment set $R_a = R_b = R_c = \{(0,0) \mapsto 0,\ (0,1) \mapsto 1,\ (1,0) \mapsto 2\}$
  - 1–ii　$R_a = R_b = R_c$（自然な Assignment set も含む）
  - 1–iii　任意の Assignment set
- $\#R_a = \#R_b = 3$，$\#R_c = 4$
  - 2–i　$R_a = R_b \subseteq R_c$
  - 2–ii　任意の Assignment set
- $\#R_a = \#R_c = 3$，$\#R_b = 4$
  - 3–i　$R_a = R_c \subseteq R_b$
  - 3–ii　任意の Assignment set
- $\#R_a = 3$，$\#R_b = \#R_c = 4$
  - 4–i　$R_a \subseteq R_b = R_c$
  - 4–ii　任意の Assignment set
- $\#R_a = \#R_b = 4$，$\#R_c = 3$
  - 5–i　$R_c \subseteq R_a = R_b$
  - 5–ii　任意の Assignment set
- $\#R_a = \#R_b = \#R_c = 4$
  - 6–i　共通の冗長表現を用いた Assignment set $R_a = R_b = R_c$
  - 6–ii　任意の Assignment set

本稿では，この Assignment set の分類を用いて GF(3) 上の加算命令列の探索を行う．

3.2 論理命令セット

多くの CPU において論理命令 AND，OR，XOR が使用可能であるが，いくつかのアーキテクチャでは，さらに異なる論理命令を使用可能な場合がある．GF(2) の元 x，y に対し，MMX や SSE による実装では ANDN（x ANDN $y = x \,\&\, \bar{y}$）が使用可能であり，また SPARC や Alpha アーキテクチャ[8),26)] では ANDN に加え ORN（x ORN $y = x \mid \bar{y}$）や XORN（x XORN $y = x \wedge \bar{y}$）が使用可能である．さらに AND，OR と NOT を組み合わせた NAND（$\overline{x \,\&\, y}$）や NOR（$\overline{x \mid y}$）が使用可能な場合がある．

探索では，使用可能な論理命令として次の 2 つの論理命令セットを扱うこととした．

LISet 8 = {AND, OR, XOR, ANDN, ORN, XORN, NAND, NOR}

LISet 3 = {AND, OR, XOR}

LISet 8 は，非可換な命令 ANDN（x ANDN $y \neq y$ ANDN x），ORN（x ORN $y \neq y$ ORN x）を含む 8 種類の論理命令から構成され，演算 $*$ に対して $x * y = x$ となるような自明な演算を除くすべての 2 項演算の結果を計算可能である．また 1 章で示した Osvik[20)] らの探索実験では NOT 命令を用いていたが，本稿では LISet 8 において NOT 命令を含む

© 2009 Information Processing Society of Japan

命令列は NOT 命令を除いた命令列に変換可能であることから NOT 命令を用いない．たとえば，$\bar{x}$ ANDN y は x NOR y と同様の結果を得る．LISet 3 は，多くの CPU で利用可能な論理命令を LISet 8 の中から選択し構成される．

3.3 探索アルゴリズム

本節では，GF(3) 上の加算 $c \leftarrow a + b$ を計算する命令列の探索方法について述べる．まずはじめに，選択した Assignment set R_a，R_b，R_c を用いた GF(3) 上の加算に対し，すべての計算パターンからなるビット列 a_H，a_L，b_H，b_L，c_H，c_L を構成する．

Assignment が冗長表現を用いるとき，GF(3) のある元が 2 個のビット表現を持つ．したがって，ビット列 a_H，a_L，b_H，b_L，c_H，c_L の長さは，入力の Assignment R_a，R_b が冗長表現を用いるかどうかにより変化する．R_a，R_b のどちらも冗長表現を用いないとき，ビット列の長さは 9 ビットとなる．また R_a，R_b の一方が冗長表現を用いるとき，ビット列の長さは 12 ビットとなり，R_a，R_b の両方が冗長表現を用いるとき，ビット列の長さは 16 ビットとなる．一方，出力の Assignment R_c が冗長表現を用いるとき，出力のビット列 c_H，c_L の冗長なビット表現を持つ部分のビットの値が一意に定まらない．よって，c_H，c_L の冗長部分のビットに対し，2 個のビット表現のどちらか一方を満たすかどうかを調べる必要がある．

表 1 に式 (3) によって構成されるビット列の例を示す．表 1 では，R_a が冗長表現を用いず，R_b が冗長表現を用いているため，ビット列の長さは 12 ビットとなる．また R_c が冗長表現を用いていないため，すべてのビット列の値は一意に定まる．

構成したビット列 a_H，a_L，b_H，b_L，c_H，c_L を用いて，深さ優先探索により GF(3) 上の加算を計算する命令列の探索を行う．N を探索を行う命令列の上限の命令数としたとき，探索アルゴリズムは次のとおりである．

(1)　探索集合 S を $S = \{a_H, a_L, b_H, b_L\}$ に初期化する．
(2)　任意の 2 個のビット列 x，y を S から選択する．

表 1　式 (3) の Assignment set R_a，R_b，R_c によるビット列の構成例
Table 1　Bit-strings for Assignment set R_a, R_b, R_c in Eq. (3).

a_H	0	0	0	0	0	0	0	0	1	1	1	1
a_L	0	0	0	0	1	1	1	1	0	0	0	0
b_H	1	0	1	0	1	0	1	0	1	0	1	0
b_L	0	1	1	0	0	1	1	0	0	1	1	0
c_H	1	0	1	1	0	1	1	1	1	1	0	0
c_L	1	1	0	0	1	0	1	1	0	1	1	1

(3)　任意の論理命令 $*$ を LISet から選択し，$z \leftarrow x * y$ を計算する．
(4)　計算結果のビット列 z を S に追加する．
(5)　Step 2–4 を上限の命令数 N まで繰り返す．
(6)　S に c_H と c_L の両方が含まれているか判別する．

探索アルゴリズムを用いてすべての命令列を計算する場合，多くの命令列は GF(3) 上の加算を計算できない命令列であるため，探索は効率的ではない．したがって，次の終了条件を満たすとき，以降の深さの探索を行わないことで探索の効率を向上させる．

- 計算結果 z がすでに S に含まれている．
- 繰返し回数が $N-1$ のとき，c_H と c_L のどちらも S に含まれていない．

計算結果 z がすでに S に含まれる場合，その命令列はより少ない命令数の命令列に変換可能である．また GF(3) 上の加算結果の命令列は $c_H \neq c_L$ であり，また GF(3) 上の加算の計算には c_H と c_L の両方が S に含まれる必要がある．したがって，繰返し回数が $N-1$ のとき，c_H または c_L の一方が S に含まれる必要がある．

3.4 探索コスト

本節では，探索実験に必要な計算コストについて述べる．

$\#R_i = 3$ である R_i のとるパターン数は，GF(3) の各元が $\mathrm{GF}(2)^2$ の異なるビット表現に割り当てられるため 24（$= 4 \times 3 \times 2$）パターンである．また $\#R_i = 4$ である R_i のとるパターン数は，さらに GF(3) のある元が $\mathrm{GF}(2)^2$ の残りのビット表現に割り当てられるため 72（$= 24 \times 3$）パターンである．したがって，Assignment set R_a，R_b，R_c において $\#R_i = 4$ となる Assignment の数を r（$0 \leq r \leq 3$）とすると，Assignment set のパターン数は $24^{3-r} \times 72^r$ となる．たとえば，$\#R_a = \#R_b = \#R_c = 3$ のとき，Assignment set のパターン数は 13,824（$= 24^3$）となる．

次に $\#R_a = \#R_b = \#R_c = 3$，$N = 5$ および LISet 8 のときの，1 つの Assignment set に対する探索に必要な計算コストを見積もる．ここで ANDN，ORN は非可換な命令のため，LISet 8 は 10 種類の論理命令として扱う．したがって，探索における論理命令の計算回数は

$$\sum_{i=1}^{5} \prod_{j=1}^{i} (10\,(j+3)(j+2)) = 1696471224120 \approx 2^{40.6}$$

となる．多くの命令列では c_H，c_L の両方を計算できないため，3.3 節で示した終了条件を適応することで探索コストを削減する．実験により，$\#R_a = \#R_b = \#R_c = 3$，$N = 5$ お

© 2009 Information Processing Society of Japan

よび LISet 8 のとき，終了条件を適応した探索での論理命令の計算回数は約 $2^{31.0}$ 回となり，終了条件を用いない場合と比較し計算回数が約 $2^{-9.6}$ となった．

3.5　探索結果と GF(3) 上の加算の構成例

探索実験では，Harrison らがすでに 7 命令での GF(3) 上の加算命令列を示しているため，7 命令以下の命令列について探索を行った．**表 2** に探索結果の詳細を示す．実験は 24 個の Pentium 4，96 個の Pentium D および 6 個の Xeon processor を用いた．$\#R_a = \#R_b = \#R_c = 3$，$N = 5$ および LISet 8 での探索実験はおよそ 1 日で完了し，また表 2 に示したすべての探索実験にはおよそ 1 カ月を要した．

本稿では，全数探索による命令列の探索実験により次の結果が得られた．

- 論理命令 {AND, OR, XOR, ANDN, ORN, XORN, NAND, NOR} や任意の Assignment set R_a，R_b，R_c を用いたとしても，5 命令以下で GF(3) 上の加算 $c \leftarrow a + b$ を計算することができない．
- $\#R_a = \#R_b = \#R_c = 3$ である Assignment set R_a，R_b，R_c および論理命令 {AND, OR, XOR} を用いて，6 命令で GF(3) 上の加算 $c \leftarrow a + b$ を計算可能である．

表 2　GF(3) 上の加算命令列の探索結果
Table 2　Search results for computing the GF(3)-addition.

$\#R_a$	$\#R_b$	$\#R_c$	Assignment set	LISet	命令数	結果
3	3	3	1-iii	8	5 以下	なし
3	3	3	1-i	3	6 以下	なし
3	3	3	1-ii	3	6	あり
3	3	4	2-ii	8	5 以下	なし
3	3	4	2-i	3	6	あり
3	4	3	3-ii	8	5 以下	なし
3	4	3	3-ii	3	6 以下	なし
3	4	3	3-i	3	7	あり
3	4	4	4-ii	8	5 以下	なし
3	4	4	4-i	3	6	あり
4	4	3	5-ii	8	5 以下	なし
4	4	3	5-ii	3	6 以下	なし
4	4	3	5-i	3	7	あり
4	4	4	6-ii	8	5 以下	なし
4	4	4	6-i	3	6	あり

LISet. LISet 3 = {AND, OR, XOR},
LISet 8 = {AND, OR, XOR, ANDN, ORN, XORN, NAND, NOR}
Assignment set. Assignment set R_a，R_b，R_c は 3.1 節に従う．

本稿に類似した命令列の探索実験として，1 章で示した Osvik [20] による 4-bit 入出力の S-box を計算する命令列の探索実験がある．本稿の LISet 8 を用いた探索では，Osvik が用いた論理命令 AND，OR，XOR に加え，論理命令 ANDN，ORN，XORN，NAND，NOR の 5 種類の命令を用いている．また NOT 命令は 3.2 節で示したとおり追加した 5 種類の論理命令により同様の命令列が構成可能である．一方，本稿では論理命令数の最小化に特化して探索を行っているため，実装において必要な MOV 命令やレジスタ数を考慮していない．

次に，新たに構成した 6 命令での GF(3) 上の加算 $c \leftarrow a + b$ の構成例をあげる．以降，GF(3) の元を $a = (a_h, a_l)$，$b = (b_h, b_l)$，$c = (c_h, c_l)$ とする．

Assignment set $R_a = R_b = R_c = \{(1,1) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}$ のとき，GF(3) 上の加算は次の命令列により計算可能となる．

$$\begin{cases} c_h & \leftarrow & (a_l \wedge b_l) \mid ((a_h \wedge b_h) \wedge a_l) \\ c_l & \leftarrow & (a_h \wedge b_h) \mid ((a_l \wedge b_l) \wedge a_h) \end{cases} \tag{4}$$

この Assignment set では，元 a に対する負の元 $-a$ は a_h と a_l の交換により計算できる．

Assignment set $R_a = R_b = R_c = \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2\}$ のとき，GF(3) 上の加算は次の命令列により計算可能となる．

$$\begin{cases} c_h & \leftarrow & (a_l \wedge b_h) \,\&\, (a_h \wedge b_l) \\ c_l & \leftarrow & (a_l \wedge b_l) \mid ((a_h \wedge b_l) \wedge b_h) \end{cases} \tag{5}$$

この Assignment set では，元 a に対する負の元 $-a$ は $a_h \leftarrow a_h \wedge a_l$ により簡単に求められるが，計算には余分な計算コストが必要となる．一方，減算 $a - b$ については次の命令列を用いることにより 6 命令で計算可能となる．

$$\begin{cases} c_h & \leftarrow & (a_h \wedge b_l) \,\&\, ((a_l \wedge b_l) \wedge b_h) \\ c_l & \leftarrow & (a_h \wedge b_h) \mid (a_l \wedge b_l) \end{cases} \tag{6}$$

さらに自然な Assignment set $R_a = R_b = R_c = \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}$ のとき，論理命令 {XOR, ANDN} を用いることで，次の命令列により 6 命令で GF(3) 上の加算が計算可能となる．

$$\begin{cases} c_h & \leftarrow & \overline{(a_l \wedge b_l)} \,\&\, ((a_h \wedge b_h) \wedge a_l) \\ c_l & \leftarrow & \overline{(a_h \wedge b_h)} \,\&\, ((a_l \wedge b_l) \wedge a_h) \end{cases} \tag{7}$$

© 2009 Information Processing Society of Japan

この命令列は入力に必要な 4 個のレジスタのみで計算可能であるため，計算結果を入力の一方に上書きする実装において，使用するレジスタ数を削減できる．式 (7) の命令列はレジスタ r_i $(i = 0, 1, 2, 3)$ に対して，次のとおりに計算する．

1. $r_0 \leftarrow a_h,\ r_1 \leftarrow a_l,\ r_2 \leftarrow b_h,\ r_3 \leftarrow b_l$
2. $r_2 \leftarrow r_0 \wedge r_2,\ r_3 \leftarrow r_1 \wedge r_3$
3. $r_1 \leftarrow r_2 \wedge r_1,\ r_0 \leftarrow r_3 \wedge r_0$
4. $r_3 \leftarrow r_1$ ANDN $r_3,\ r_2 \leftarrow r_0$ ANDN r_2
5. $c_h \leftarrow r_3,\ c_l \leftarrow r_2$

4. $\mathbf{GF}(3^m)$ 上の η_T ペアリングへの適応

本章では，前章までで示した GF(3) 上の加算を用いて GF(3^m) 上の演算および η_T ペアリングの実装，評価を行う．実装は GCC 4.1.2 コンパイラ，`-O3` オプションにより行い，評価は AMD Opteron model 275（2.2 GHz），Linux/x86_64 上で行った．AMD Opteron model 275 のキャッシュサイズは L1 キャッシュ（データ/命令）が 64 KByte/64 KByte，L2 キャッシュが 1 MByte である．計算時間の測定には，時間の測定に `timeval` 構造体および `gettimeofday` 関数を，クロックの測定にアセンブリ言語の `rdtsc` 命令を用いた．また実装において SSE などの拡張命令は使用しておらず，論理命令 ANDN を用いることができない．

4.1 $\mathbf{GF}(3^m)$ の元のビット表現

GF(3)[x] を GF(3) 上のすべての多項式の集合，$f(x)$ を GF(3) 上の既約多項式としたとき，標数 3 の有限体 GF(3^m) は GF(3)[x]/($f(x)$) と定義される．また GF(3^m) の元 A は多項式表現 $\sum_{i=0}^{m-1} a_i x^i$ $(a_i \in \mathrm{GF}(3))$ により表現される．GF(3^m) の元 A の各係数 a_i はビット表現 $a_i = ((a_i)_h, (a_i)_l) \in \mathrm{GF}(2)^2$ を用いて 2 ビットで表現し，元 $A \in \mathrm{GF}(3^m)$ は bit-slice 表現[22] を用いて次の 2 個のビット列 (A_h, A_l) により表現する．

$A_h = ((a_{m-1})_h, (a_{m-2})_h, \ldots, (a_0)_h)$

$A_l = ((a_{m-1})_l, (a_{m-2})_l, \ldots, (a_0)_l)$

このビット列は実装対象のワード長 W に対して，サイズ $\lceil m/W \rceil$ の配列 2 個を用いて保存する．本稿では，ワード長を $W = 64$ として実装する．

ここで係数 $a_i \in \mathrm{GF}(3)$ の $\mathrm{GF}(2)^2$ への割当てとして次の 3 つの Assignment を用いる．1 つは Harrison らが用いている自然な Assignment，残りの 2 つは 3.5 節で示した 6 命令で GF(3) 上の加算が計算可能な Assignment である．用いる Assignment は次のとおりである．

$$\begin{cases} \textit{Natural}\ \text{assignment} : \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\} \\ \textit{Type 1}\ \text{assignment} : \{(1,1) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\} \\ \textit{Type 2}\ \text{assignment} : \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2\} \end{cases}$$

次節より，これらの Assignment を用いた GF(3^m) 上の加算，減算，乗算および η_T ペアリングの計算時間の評価を行う．

4.2 $\mathbf{GF}(3^m)$ 上の加算および減算

多項式表現の元 A，B，$C \in \mathrm{GF}(3^m)$ に対し，GF(3^m) 上の加算 $C \leftarrow A + B$ は元 A，B の各係数に対する GF(3) 上の加算 $c_i \leftarrow a_i + b_i$ により計算される．*Natural*，*Type 1*，*Type 2* assignment における GF(3) 上の加算命令列はそれぞれ式 (2)，(4)，(5) を用いる．実装では bit-slice 表現を用いることにより 1 回の論理命令で W 個のビットを同時に計算可能であるため，GF(3^m) 上の加算は *Natural* assignment では $7\lceil m/W \rceil$ 回，また *Type 1*，*Type 2* assignment では $6\lceil m/W \rceil$ 回の論理命令により計算することができる．

同様に，GF(3^m) 上の減算 $C \leftarrow A - B$ も論理命令を用いて構成する．*Natural* および *Type 1* assignment では，元 B に対する負の元 $-B$ を B_h と B_l の交換により計算できるため，$A + (-B)$ により加算と同様の計算コストで減算が計算可能である．一方，*Type 2* assignment では，式 (6) を用いることにより $6\lceil m/W \rceil$ 回の論理命令で減算が計算可能である．

本稿では，拡大体 GF($3^{6 \cdot 509}$) 上の離散対数問題が 128-bit セキュリティを持つと見積もられているため[1),18)]，計算時間の評価に有限体 GF(3^{509}) を用いる．**表 3** に GF(3^{509}) 上の加算および減算の計算時間の結果を示す．計算時間の測定は，100,000,000 回の実行に対する平均により行っている．

理論的な見積りでは，GF(3) 上の加減算で用いる論理命令数が 7 から 6 に減少したため，*Type 1* および *Type 2* assignment を用いた加減算の計算時間は，*Natural* assignment を用いた場合と比較し 6/7 となる．一方，実装結果では，GF(3^m) 上の加減算の両方において *Type 1*，*Type 2* assignment を用いた加減算は *Natural* assignment を用いた場合に比べ 11〜16%（$\simeq 1/7$）高速となった．これは GF(3^m) 上の加減算がシンプルな演算であり，$6\lceil m/W \rceil$ または $7\lceil m/W \rceil$ 回の論理命令以外の計算コストをほとんど必要としないためである．したがって，計算に必要な論理命令数を減少させることにより，GF(3^m) 上の加減算

© 2009 Information Processing Society of Japan

表 3　GF(3^{509}) 上の加算および減算の計算時間（Time: μsec，Clock: cycles）
Table 3　Running time comparison for addition and subtraction in GF(3^{509}) (Time: μsec, Clock: cycles).

		Natural	*Type 1*	*Type 2*
加算	Time	0.0197	0.0166	0.0175
	Clock	44	37	39
減算	Time	0.0199	0.0166	0.0176
	Clock	44	37	39

表 4　GF(3^{509}) 上の乗算の計算時間（Time: μsec，Clock: cycles）
Table 4　Running time comparison for multiplication in GF(3^{509}) (Time: μsec, Clock: cycles).

		Natural	*Type 1*	*Type 2*
Shift-and-add	Time	7.09	6.91	6.62
($w = 4$)	Clock	15.7×10^3	15.2×10^3	14.7×10^3
Comb	Time	4.98	4.74	4.59
($w = 4$)	Clock	11.0×10^3	10.5×10^3	10.1×10^3

の計算時間は理論的な見積りと同程度高速となった．

4.3　GF(3^m) 上の乗算

元 A，B，$C \in$ GF(3^m) および GF(3^m) の既約多項式 $f(x)$ に対し，GF(3^m) 上の乗算 $A \cdot B$ は GF(3)[x] 上の多項式乗算 $C' \leftarrow A \cdot B$ およびリダクション $C \leftarrow C' \bmod f(x)$ により構成される．$f(x)$ として既約 3 項式 $f(x) = x^m + x^k + 2$ が存在するとき，リダクションの計算コストは多項式乗算と比べとても小さい[14]．したがって，以降，多項式乗算の効率的なアルゴリズムについて述べる．

多項式乗算は GF(3^m) 上の加算およびシフト演算 Ax^s（$s \in \mathbb{N}$）により計算される．シフト演算 Ax^s は入力 s，$A = \sum_{i=0}^{m-1} a_i x^i$ より $\sum_{i=0}^{m-1} a_i x^{i+s}$ を求める関数である．多項式乗算の最も簡単なアルゴリズムとして Shift-and-add 法[14] がある．Shift-and-add 法では，i が 0 から $m-1$ まで $C' \leftarrow C' + Ab_i x^i$ を実行する．また Comb 法では，i が 0 から W まで，$C' \leftarrow C' + Ab_i x^i$ の計算後に $C' \leftarrow C' + Ab_{jW+i} x^{jW+i}$（$0 < j < \lceil m/W \rceil$）を実行する．$Ax^i$ の結果を用いることでワード長のシフト演算 Ax^{jW+i} が計算コストを必要としないことから，Comb 法は Shift-and-add 法に比べ高速に計算可能である．

さらに Shift-and-add 法や Comb 法で使用可能な効率的な計算アルゴリズムである Window 法[14] を適応する．Window 法は事前計算を行うことで多項式乗算における GF(3^m) 上の加算およびシフト演算の回数を削減するアルゴリズムである．Window 幅 w に対して，事前計算に GF(3^m) 上の加算 $(3^w - 2w - 1)/2$ 回とシフト演算 $w - 1$ 回が必要であり，また多項式乗算の計算時間は Window 法を用いない場合と比較し約 $1/w$ となる．

本稿では，Window 幅 $w = 4$ の Shift-and-add 法および Comb 法の実装を行った．**表 4** に GF(3^{509}) 上の乗算の計算時間を示す．実装では，GF(3^{509}) の多項式基底として $f(x) = x^{509} + x^{358} + 2$ を用いた．計算時間の測定は，1,000,000 回の実行に対する平均により行っている．

実装では，シフト演算 Ax^s の際，x^s よりも小さな次数の係数のビットには $0 \in$ GF(2) がパディングされる．*Type 1* assignment では，$0 \in$ GF(3) は $(1,1) \in$ GF$(2)^2$ に割り当てられているため，パディングされたビットを 0 から 1 に変換する必要がある．また *Type 2* assignment では，負の元の計算 $A_h \leftarrow A_h \wedge A_l$ に $\lceil m/W \rceil$ 回の論理命令が必要となる．GF(3^{509}) 上の乗算では，計算コストの大部分が加算とシフト演算であるため，GF(3^{509}) 上の加算での結果とは異なり，*Type 2* assignment を用いた乗算が *Type 1* assignment を用いた場合と比較し数%高速となった．また *Type 2* assignment と *Natural* assignment の比較では，*Type 2* assignment を用いることにより Window 幅 $w = 4$ の Shift-and-add 法において約 7%，Window 幅 $w = 4$ の Comb 法において約 8%高速となった．結果，*Type 2* assignment を用いた Window 幅 $w = 4$ の Comb 法による乗算が最も高速であった．

4.4　GF(3^m) 上の η_T ペアリング

GF(3^m) 上の η_T ペアリングは Barreto らによって提案された[3]．また η_T ペアリングを効率的に計算するために，これまで様々な高速化アルゴリズムが提案されている．本稿では，Beuchat らが提案したループ展開法[5]，Gorla が提案した GF(3^{6m}) 上の高速乗算[11]，Shirase らが提案した 3 乗根を用いない η_T ペアリング[24] および GF(3^m) 上のトーラス T_2 を用いた効率的な最終べき[25]，Takahashi らが提案した Window 法を用いた GF(3^m) 上の乗算での事前計算テーブル再利用法[27] を用いて η_T ペアリングを実装した．

GF(3^m) 上の演算の実装には，乗算に Window 幅 $w = 4$ の Comb 法，3 乗算に入力 $\sum_{i=t}^{t+10} a_i x^i$ に対し $\sum_{i=t}^{t+10} a_i x^{3i}$ を計算する 11-bit テーブルおよび既約 3 項式でのリダクション，また逆元算に 3 進の多項式における拡張ユークリッド互除法[14] を用いた．3 乗算で用いる 11-bit テーブルのサイズは 32×2^{11} bit = 8 KByte であり，L1 キャッシュに収まるテーブルサイズである．

表 5 に GF(3^{509}) 上の η_T ペアリングの計算時間を示す．計算時間の測定は，100,000 回の実行に対する平均により行っている．

η_T ペアリングの計算コストのほとんどが GF(3^m) 上の乗算であることから，*Type 2* as-

© 2009 Information Processing Society of Japan

表 5　GF(3^{509}) 上の η_T ペアリングの計算時間（Time: μsec，Clock: cycles）
Table 5　Running time comparison for η_T pairing over GF(3^{509}) (Time: μsec, Clock: cycles).

		Natural	*Type 1*	*Type 2*
η_T ペアリング	Time	17,525	17,238	16,295
	Clock	38.7×10^6	37.6×10^6	36.1×10^6

signment を用いた η_T ペアリングは *Natural* assignment を用いた場合と比較し，Window 幅 $w = 4$ の Comb 法による乗算と同程度の約 7%高速となった．一方，*Type 1* assignment を用いた η_T ペアリングは，3 乗算や逆元算において多くのシフト演算を必要するため，*Natural* assignment を用いた場合と比べ 2%程度の高速化にとどまった．

結果として，*Type 2* assignment を用いた GF(3^{509}) 上の η_T ペアリングが最も高速となり，計算時間は 16.3 msec であった．

最後に，最も高速であった *Type 2* assignment を用いて，様々な次数における GF(3^m) 上の演算および η_T ペアリングの計算時間を示す．用いた次数と既約 3 項式 $f(x) = x^m + x^k + 2$ はそれぞれ $(m, k) = \{(97, 12), (167, 96), (193, 12), (239, 24), (353, 142), (509, 358)\}$ である．Page ら[21] は次数 $m = 97$，163，193，239，353 における GF(3^{6m}) の離散対数問題がそれぞれ RSA 845，912，1080，1338，1976-bit セキュリティを持つと見積もっている．**表 6** に *Type 2* assignment を用いた GF(3^m) 上の演算および η_T ペアリングの計算時間を示す．

比較のため，近年発表された GF(3^m) 上の η_T ペアリングの高速実装結果を**表 7** に示す．Ahmadi ら[1] は GF(3^{509}) 上の加算が 0.09 μsec，Window 幅 $w = 3$ の Comb 法を用いた乗算が 15.5 μsec と示している．彼らは既約多項式に $f(x) = x^{509} - x^{477} + x^{445} + x^{32} - 1$，実装には GCC 3.3 コンパイラを用い，Pentium 4（2.4 GHz），Linux/x86 上で評価を行っている．Hankerson ら[13] は様々な実装環境において実装を行っており，64-bit Opteron（2.8 GHz）上で SSE や MMX を使用しない実装において，GF(3^{509}) 上の乗算が 10.6×10^3 cycles（約 3.79 μsec@2.8 GHz），η_T ペアリングが 46×10^6 cycles（約 16.4 msec@2.8 GHz）と示している．彼らは既約多項式に $f(x) = x^{509} - x^{318} - x^{191} + x^{127} + 1$，また実装には GCC 4.1 コンパイラを用いている．Beuchat ら[6] は SSE を用いた高速実装を行い，マルチコア環境での評価を行っており，GF(3^{509}) 上の η_T ペアリングは 7.96 msec（1 コア），4.53 msec（2 コア），3.28 msec（4 コア）を達成している．彼らは既約多項式に $f(x) = x^{509} - x^{318} - x^{191} + x^{127} + 1$ を用いている．また実装は Visual Studio 2008 SP1 コンパイラを用い，Core 2（2.4 GHz），Windows XP 64-bit SP2 上で動作させている．彼らは GF(3) 上の加算において，著者ら

表 6　*Type 2* assignment を用いた GF(3^m) 上の演算および η_T ペアリングの計算時間（Time: μsec，Clock: cycles）
Table 6　Running times for arithmetic in GF(3^m) and the η_T pairing with the *Type 2* assignment (Time: μsec, Clock: cycles).

次数 m		97	167	193	239	353	509
加算	Time	0.0057	0.0075	0.0098	0.0098	0.0137	0.0175
	Clock	13	17	22	22	30	39
減算	Time	0.0057	0.0075	0.0098	0.0098	0.0135	0.0176
	Clock	13	17	22	22	30	39
乗算 [1]	Time	0.77	1.37	1.82	1.82	3.31	4.59
	Clock	1.7×10^3	3.0×10^3	4.0×10^3	4.0×10^3	7.2×10^3	10.1×10^3
3 乗算	Time	0.073	0.096	0.122	0.130	0.182	0.282
逆元算	Time	6.9	13.7	19.2	23.8	49.4	94.3
η_T ペアリング [2]	Time	615	1,688	2,611	3,157	8,299	16,295
	Clock	1.4×10^6	3.7×10^6	5.7×10^6	7.0×10^6	17.9×10^6	36.1×10^6

[1] Window 幅 $w = 4$ の Comb 法[14]．
[2] ループ展開法[5]，GF(3^{6m}) 上の高速乗算[11]，3 乗根を用いない η_T ペアリング[24]，トーラス T_2 を用いた最終べき[25]，乗算の事前計算テーブル再利用法[27]．

表 7　GF(3^{509}) 上の乗算および η_T ペアリングにおける従来の高速実装結果（Time: μsec，Clock: cycles）
Table 7　Recently reported running times for multiplication and the η_T pairing over GF(3^{509}) (Time: μsec, Clock: cycles).

著者	CPU	演算	計算時間	
Ahmadi ら[1]	Pentium 4 (2.4 GHz, 32-bit)	乗算	Time	15.5
Hankerson ら[13]	Opteron (2.8 GHz, 64-bit)	乗算	Clock	10.6×10^3
		η_T ペアリング	Clock	46×10^6
Beuchat ら[6]	Core 2 (2.4 GHz, 64-bit, SSE)	乗算	Time	1.8
		η_T ペアリング	Time	7.96×10^3

が文献 17) において示した 6 命令での命令列よりも 7 命令である式 (2) の命令列を用いた実装が高速であるという結果を示している．したがって，実装環境によって使用する命令列の適切な選択が重要である．

5. ま と め

本稿では，GF(3) 上の加算を計算するため論理命令を用いた命令列の探索を行った．探索実験では，GF(3) の元の冗長表現や加算写像 GF(3) × GF(3) → GF(3) の各 GF(3) に対し

© 2009 Information Processing Society of Japan

て任意の $GF(2)^2$ への割当てを行った．また使用可能な論理命令として，多くの CPU で利用可能な論理命令 {AND, OR, XOR} に加え，SPARC や Alpha アーキテクチャ，MMX, SSE などの実装において使用可能な論理命令 {ANDN, ORN, XORN, NAND, NOR} を用いた．

探索結果として，任意の論理命令や GF(3) の元の $GF(2)^2$ へのビット割当てを用いたとしても 5 命令以下で GF(3) 上の加算は計算できず，また $\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2\}$ などの自然ではないビット割当て，または {AND, OR, XOR} 以外の新たな論理命令を用いることにより，6 命令で GF(3) 上の加算を計算可能な命令列を構成した．

最後に，新たに構成した 6 命令での GF(3) 上の加算を用いて，$GF(3^m)$ 上の演算および η_T ペアリングのソフトウェア実装を行った．$GF(3^{509})$ 上の η_T ペアリングの計算時間は AMD Opteron model 275（2.2 GHz）上で約 16.3 msec であり，従来の 7 命令での GF(3) 上の加算を用いた場合と比較し約 7%高速となった．

参 考 文 献

1) Ahmadi, O., Hankerson, D. and Menezes, A.: Software implementation of arithmetic in $\mathbb{F}_{3^m}$, *Proc. WAIFI 2007*, LNCS, Vol.4547, pp.85–102 (2007).
2) Barker, E., Barker, W., Burr, W., Polk, W. and Smid, M.: Recommendation for Key Management – Part 1: General (Revised), NIST Special Publication 800-57 (2007).
3) Barreto, P., Galbraith, S., Ó'hÉigeartaigh, C. and Scott, M.: Efficient pairing computation on supersingular abelian varieties, *Designs, Codes and Cryptography*, Vol.42, No.3, pp.239–271 (2007).
4) Bertoni, G., Guajardo, J., Kumar, S., Orland, G., Paar, C. and Wollinger, T.: Efficient $GF(p^m)$ arithmetic architectures for cryptographic applications, *Proc. CT-RSA 2003*, LNCS, Vol.2612, pp.158–175 (2003).
5) Beuchat, J., Brisebarre, N., Detrey, J., Okamoto, E., Shirase, M. and Takagi, T.: Algorithms and arithmetic operators for computing the η_T pairing in characteristic three, Cryptology ePrint Archive, Report 2007/417 (2007).
6) Beuchat, J., López-Trejo, E., Martínez-Ramos, L., Mitsunari, S. and Rodríguez-Henríquez, F.: Multicore implementation of the Tate pairing over supersingular elliptic curves, Cryptology ePrint Archive, Report 2009/276 (2009).
7) Boneh, D. and Franklin, M.: Identity based encryption from the Weil pairing, *SIAM J. Comput.*, Vol.32, No.3, pp.586–615 (2003).
8) Compaq Computer Corporation: Alpha Architecture Handbook (Version 4) (1998).
9) Duursma, I. and Lee, H.: Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$, *Proc. ASIACRYPT 2003*, LNCS, Vol.2894, pp.111–123 (2003).
10) Galbraith, S., Harrison, K. and Soldera, D.: Implementing the Tate pairing, *Proc. ANTS 2002*, LNCS, Vol.2369, pp.324–337 (2002).
11) Gorla, E., Puttmann, C. and Shokrollahi, J.: Explicit formulas for efficient multiplication in $\mathbb{F}_{3^{6m}}$, *Proc. SAC 2007*, LNCS, Vol.4876, pp.173–183 (2007).
12) Granger, R., Page, D. and Stam, M.: Hardware and software normal basis arithmetic for pairing-based cryptography in characteristic three, *IEEE Trans. Computers*, Vol.54, No.7, pp.852–860 (2005).
13) Hankerson, D., Menezes, A. and Scott, M.: Software implementation of pairings, Centre for Applied Cryptographic Research (CACR) Technical Reports, CACR 2008-08 (2008). available at http://www.cacr.math.uwaterloo.ca/techreports/2008/cacr2008-08.pdf
14) Hankerson, D., Menezes, A. and Vanstone, S.: *Guide to Elliptic Curve Cryptography*, Springer (2004).
15) Harrison, K., Page, D. and Smart, N.: Software implementation of finite fields of characteristic three, for use in pairing-based cryptosystems, *LMS Journal of Computation and Mathematics*, Vol.5, pp.181–193 (2002).
16) Intel Corporation: Intel Architecture Software Developer's Manual (Volume 2: Instruction Set Reference) (1999).
17) Kawahara, Y., Aoki, K. and Takagi, T.: Implementation of η_T pairing over $GF(3^m)$ using minimum number of logical instructions for GF(3)-addition, *Proc. Pairing 2008*, LNCS, Vol.5209, pp.282–296 (2008).
18) Koblitz, N. and Menezes, A.: Pairing-based cryptography at high security levels, *Proc. Cryptography and Coding 2005*, LNCS, Vol.3796, pp.13–36 (2005).
19) Miller, V.: Short program for functions on curves, Unpublished manuscript (1986).
20) Osvik, D.: Speeding up Serpent, 3rd AES Candidate Conference (AES3) (2000). available at http://www.ii.uib.no/osvik/pub/aes3.pdf
21) Page, D., Smart, N. and Vercauteren, F.: A comparison of MNT curves and supersingular curves, *Applicable Algebra in Engineering, Communication and Computing*, Vol.17, No.5, pp.379–392 (2006).
22) Page, D. and Smart, N.: Hardware implementation of finite fields of characteristic three, *Proc. CHES 2002*, LNCS, Vol.2523, pp.529–539 (2003).
23) Sakai, R. and Kasahara, M.: ID based cryptosystems with pairing on elliptic curve, Cryptology ePrint Archive, Report 2003/054 (2003).
24) Shirase, M., Kawahara, Y., Takagi, T. and Okamoto, E.: Universal η_T pairing algorithm over arbitrary extension degree, *Proc. WISA 2007*, LNCS, Vol.4867, pp.1–15 (2008).

© 2009 Information Processing Society of Japan

25) Shirase, M., Takagi, T. and Okamoto, E.: Some efficient algorithms for the final exponentiation of η_T pairing, *Proc. ISPEC 2007*, LNCS, Vol.4464, pp.254–268 (2007).
26) SPARC International, Inc.: The SPARC Architecture Manual, Version 9 (2000).
27) Takahashi, G., Hoshino, F. and Kobayashi, T.: Efficient GF(3^m) multiplication algorithm for ηT pairing, Cryptology ePrint Archive, Report 2007/463 (2007).

(平成 21 年 6 月 17 日受付)
(平成 21 年 9 月 11 日採録)

川原　祐人（学生会員）

2007 年公立はこだて未来大学システム情報科学部卒業．2009 年同大学大学院システム情報科学研究科博士（前期）課程修了．現在，同大学大学院システム情報科学研究科博士（後期）課程在学中．平成 21 年度より日本学術振興会特別研究員 DC1．平成 19 年度情報処理学会山下記念研究賞受賞．暗号および情報セキュリティに関する研究に従事．電子情報通信学会会員．

青木和麻呂

1993 年早稲田大学理工学部数学科卒業．1995 年同大学大学院理工学研究科修士課程修了．同年日本電信電話株式会社入社．2001 年博士（理学）（早稲田大学）．現在，NTT 情報流通プラットフォーム研究所主任研究員．SCIS'95，SCIS'96 論文賞受賞，平成 9 年度学術奨励賞（電子情報通信学会）受賞，平成 17 年度業績賞（情報処理学会）受賞，2007 年 IWSEC BEST PAPER AWARD 受賞．2007 年から早稲田大学非常勤講師，2009 年名古屋大学招へい教員．電子情報通信学会，IACR 各会員．

高木　　剛（正会員）

1993 年名古屋大学理学部数学科卒業．1995 年同大学大学院理学研究科修士課程修了．同年日本電信電話株式会社入社．2001 年理学博士（ダルムシュタット工科大学）．その後，ダルムシュタット工科大学情報科学部助教授を経て，2005 年公立はこだて未来大学システム情報科学部准教授，2008 年より同大学教授，現在に至る．2009 年より九州大学大学院数理学研究院客員教授．第 8 回船井情報科学振興賞受賞．暗号および情報セキュリティに関する研究に従事．電子情報通信学会，IACR 各会員．

© 2009 Information Processing Society of Japan