情報処理学会論文誌 Vol. 49 No. 11 1–11 (Nov. 2008)

ATmega128L上でのペアリング暗号の高速実装

石黒 司[†1,*1] 白勢政明[†1] 高木 剛[†1]

近年, ユビキタス社会の発達に伴い, 無線センサネットワーク (WSN) が実用化され始めており, 鍵共有問題やプライバシーの保守などのセキュリティ問題が議論されている. 一方, 従来の公開鍵暗号では構築が困難であった新たな暗号アプリケーションを実現できる次世代の公開鍵暗号方式として, ペアリング暗号が注目されている. センサノードとして広く研究されている MICAz は, CPU が ATmega128L であり, ワード長が 8 ビット, クロック周波数は 7.37MHz である.

Oliveira らによって実装された, 有限体 $\mathbb{F}_q(q = p^2, p$ は 256 ビットの素数) を用いた Tate ペアリングは, MICAz 上で約 30sec の時間が必要である. 本稿では MICAz 上で TinyOS を用いて標数 3 の体 $\mathbb{F}_{3^{97}}$ 上での η_T ペアリングの実装評価を行った. 高速化にあたり ATmega128L に特化した有限体の乗算や既約多項式の選択, 有限体の演算回数の削減を行った. 新たな高速化手法として, 事前計算による並び替えテーブルを用いた有限体の 3 乗算の高速化, 最小の加算回数を有する既約 3 項式による乗算及び 3 乗算のリダクションの高速化を提案した. その結果, η_T ペアリングの計算時間が約 5.8sec に改善された. また, 拡大次数 $m = 167, 193, 239$ においても実装を行い速度を評価した.

Efficient Implementation of the Pairing on ATmega128L

Tsukasa ISHIGURO,[†1,*1] Masaaki SHIRASE[†1] and Tsuyoshi TAKAGI[†1]

The technology of wireless sensor network (WSN) has been implemented in practical applications of ubiquitous society. However, the problems of security in WSN have also been discussed, for instances, key establishment, trust setup, privacy issue and so on. One of the methods for solving them is to use a public key cryptosystem. Especially, pairing based cryptosystems can achieve novel cryptographic applications such as efficient broadcast encryption. A platform fluently used for the research of sensor network is MICAz which equips an 8-bit CPU ATmega128L at 7.37MHz. Oliveira et al. implemented the Tate pairing of a supersingular elliptic curve over $\mathbb{F}_q$ ($q = p^2$ with 256-bit prime p) on ATmega128L, whose timing is about 30 seconds. In this paper, we evaluated an efficient implementation of η_T pairing over finite field $\mathbb{F}_{3^{97}}$ on ATmega128L. In order to achieve a more efficient implementation we deploy an 8-bit Comb method, an efficient reduction trinomials (ROTs), and an improved multiplication on extension field $\mathbb{F}_{(3^{97})^6}$. Indeed we proposed a faster cubing with a pre-computed table with ROTs optimized for ATmega128L. Then our implementation achieved about 5.8 seconds for computing η_T pairing over $\mathbb{F}_{3^{97}}$. We also presented and evaluated the pairing with extension degrees 167, 193, and 239.

1. はじめに

近年, センサネットワークという技術が実用化されつつある. その中でも無線センサネットワーク (WSN) の技術が様々なところで研究され, 今後本格的に実用化されるといわれている[21].

センサネットワークでは新たなセキュリティの問題が議論されている. Perrig らによると, 新たな問題として, 複数のノード同士による鍵共有, 機密性と認証などがあげられている. また, これらを解決する手段として公開鍵暗号方式が適しているが, センサネットワークを構成するセンサノードの計算資源の乏しさから実現しにくいと指摘している[18]. そのため公開鍵暗号をセンサノードに特化し, 高速実装することが重要である.

一方, 次世代の公開鍵暗号方式として, ペアリング暗号が注目されている. ペアリング暗号を利用すると, 暗号文のサイズが受信者数によらず一定になるブロードキャスト暗号などセンサネットワークに有効な暗号アプリケーションが実現できるようになる[2].

現在, センサネットワークの研究用プラットフォームとして MOTE が最も利用されている[13]. MOTE で使われている無線通信デバイスとして代表的なものに MICAz がある. 従って, 本稿では MICAz で使われている CPU である ATmega128L を実装対象とした. ATmega128L で暗号を実装するには, TinyOS[22] と呼ばれる基本ソフト上にアプリケーションを実装する方法と, TinyOS を使わずに ATmega128L 上にアセンブリのみを用いて実装する方法がある. TinyOS 上に実装される方法に比べ, アセンブリ実装の方が機械語に近いレベルで最適化することができるため, より高速な実装が可能である. TinyOS 上の実装では NesC 言語と呼ばれる C 言語を拡張した言語を使用する. アセンブリ実装は, NesC 言語

†1 公立はこだて未来大学システム情報科学部
School of Systems Information Science, Future University-Hakodate
*1 現在, 情報セキュリティ大学院大学情報セキュリティ研究科
Presently with Graduate School of Information Security, Institute of Information Security

での実装に比べると高速となるが, TinyOS に実装されている機能が使用不可となる. そのため, 本稿では TinyOS 上に NesC 言語を用いて実装を行った.

ATmega128L 上に RSA 暗号や楕円曲線暗号, Tate ペアリングが実装されてきた. Gura らはアセンブリによる実装で 1024 ビットの RSA 暗号が 0.43sec, 標数 2 の有限体である $\mathbb{F}_{2^m}$ を用いた 160 ビットの楕円曲線暗号が 0.81sec という結果を示した[9]. また, Liu らは TinyOS 上での楕円曲線暗号である TinyECC を実装した. TinyECC では奇標数の有限体である $\mathbb{F}_q$ (q は大きな素数) が用いられ, 160 ビットの楕円曲線暗号が約 1.9sec で実装されている[12]. TinyECC はコードの要所部分だけをアセンブリで記述するインラインアセンブリと呼ばれる機能を利用し高速化を行っている. Gura らのアセンブリ実装と比較すると, 実行時間が約 2 倍となっている. また, Oliveira らは TinyOS 上での Tate ペアリングである TinyTate を実装した. TinyTate は有限体 $\mathbb{F}_q(q = p^2, p$ は 256 ビットの素数) 上で定義された楕円曲線上の Tate ペアリングであり, 約 30sec の計算時間が必要である[16]. 最近になり, 標数 2 の η_T ペアリングを ATmega128L で実装した TinyPBC が発表された. TinyPBC は約 1024 ビットのセキュリティである $\mathbb{F}_{2^{271}}$ 上において 5.5sec の速度となっている[17]. しかし, 標数 3 の η_T ペアリングを ATmega128L において実装した報告はない. 本稿では, $\mathbb{F}_{3^m}$ 上の η_T ペアリングの実装報告を行う.

現在, 高速に動作するペアリング暗号として標数 3 の超特異曲線を用いた η_T ペアリングがある[1),3)]. η_T ペアリングは標数 2, または 3 の超特異曲線上のみで定義される高速なペアリングである. 標数 2, または 3 の η_T ペアリングの埋め込み次数は, それぞれ 4, または 6 となる. そのため, 標数 3 の η_T ペアリングを利用した方が, 有限体のサイズを小さくできるという長所を持っている[1]. 本稿では MICAz 上で TinyOS を用いて標数 3 の体 $\mathbb{F}_{3^m}$ 上での η_T ペアリングの実装を行った. 本稿の実装では, RSA の 1024 ビットとほぼ同等なセキュリティを実現するため, $m = 97$ を採用した.

高速化にあたり ATmega128L に特化した有限体の乗算, 3 乗算の高速化, 有限体の演算回数の削減を行った. 有限体乗算は多項式乗算とリダクションから構成される. 多項式乗算のアルゴリズムとしては Shift Add 法や Comb 法, Window 法などがある[10]. これらを実装し, その中で一番高速であった改良 Comb 法を選択した.

また, 有限体の演算を削減するために, Gorla らによって示された 6 次拡大体の乗算[6], Beuchat らによって示された η_T ペアリングのメインループのアルゴリズムを適用した[4].

本稿では, さらに以下の新規な手法を取り入れた. 有限体の 3 乗算は事前計算を用いた並び替えテーブルを使うアルゴリズムを提案し, 高速化を行った. また, 中間項の次数がワード長の整数倍となる既約 3 項式 ROTs: $f(x) = x^{97} + x^{16} + 2$ を選択することにより, リダクションを高速化した[15]. さらに, ROTs のうちで, 有限体の加算回数が最小になるように選択を行うことにより 3 乗算を高速化した.

初期実装では η_T ペアリングの計算時間は約 30sec であったが, 上記の有限体の演算の高速化を行うことによって約 8sec まで高速化された. さらに, Gorla らの拡大体乗算法, 及び Beuchat らの改良 η_T ペアリングのアルゴリズムの適用により, η_T ペアリングの計算時間を約 5.8sec に改善することができた. 加えて, 拡大次数 $m = 167, 193, 239$ においても実装を行った.

2. ATmega128L での従来のペアリング実装

本節では ATmega128L 上にペアリングを実装した TinyTate[16] を説明する. TinyTate は 2007 年に Oliveira らによって実装された MICAz 上で動作する Tate ペアリングである. Tate ペアリングの定義を以下に説明する.

$E(\mathbb{F}_{q^k})$ を体 $\mathbb{F}_{q^k}$ 上の楕円曲線の点の集合とする. ここで q を素数 p の冪, r を $r|\#E(\mathbb{F}_q)$ を満たす大きな素数, k を $r|(q^k - 1)$ を満たす最小の自然数とする. k は埋め込み次数と呼ばれる. また $E(\mathbb{F}_q)[r]$ を体 $\mathbb{F}_q$ 上の楕円曲線で位数 r の点の部分群とする. Tate ペアリングの定義は以下のような写像である.

$$\langle \cdot, \cdot \rangle : E(\mathbb{F}_q)[r] \times E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) \to \mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$$

Tate ペアリングは $P, Q \in E(\mathbb{F}_{3^m})[r], a \in \mathbb{Z}$ に対し, 双線形性

$$\langle aP, Q \rangle = \langle P, aQ \rangle = \langle P, Q \rangle^a$$

を満たす.

TinyTate は有限体は $\mathbb{F}_q, (q = p^2, p$ は 256 ビットの素数) を用い, $k = 2$, r を 128 ビットの素数を利用している. また, 楕円曲線は $E/\mathbb{F}_q : y^2 = x^3 + x$ を選択している. TinyTate では Miller のアルゴリズム[14] を用いて Tate ペアリングを計算している. TinyTate は MICAz 上に TinyOS を用いて NesC 言語により実装された.

表 1　ATmega128L のスペック
Table 1　Specification of ATmega128L

クロック周波数	7.37MHz
ワード長	8 ビット
ROM	128kB
RAM	4kB

MICAz は, クロック周波数 7.37MHz の 8 ビット CPU である ATmega128L を使用している．ATmega128L は 1 つの ALU と 32 個の汎用の 8 ビットレジスタ, データ用メモリとプログラム用メモリを含む 8 ビット CPU である．ALU では除算を除く基本的な整数演算と論理演算がサポートされており, ほとんどの命令が 1 クロックサイクル (乗算やデータメモリへのアクセスの一部は 2 クロックサイクル) でなされる．また, プログラム格納領域 (ROM) が 128kB, メモリ (RAM) が 4kB となっている．

TinyTate による Tate ペアリングの計算時間は 30.21sec である．また RAM は 4kB 中約 1.8kB 使用しており, ROM は 128kB 中約 18kB 使用している．

3. η_T ペアリングの実装方法

本節では標数 3 の体上の超特異曲線を利用した η_T ペアリング[1),3)] の説明と, η_T ペアリングのアルゴリズムを説明をする[4)]．また, それらを実装する際の標数 3 の有限体 $\mathbb{F}_{3^m}$, 3 次拡大体 $\mathbb{F}_{3^{3m}}$, 6 次拡大体 $\mathbb{F}_{3^{6m}}$ の演算について説明する．

3.1 η_T ペアリングの定義

本節では η_T ペアリングの定義について説明する．η_T ペアリングは標数 3 の場合は超特異曲線 $y^2 = x^3 - x + b, b = \pm 1$ 上で定義される．$E(\mathbb{F}_{3^m})$ の位数 r の部分群を $E(\mathbb{F}_{3^m})[r]$ とすると, η_T ペアリングは以下のような写像である．

$$\eta_T : E(\mathbb{F}_{3^m})[r] \times E(\mathbb{F}_{3^{6m}})[r] \to \mathbb{F}^*_{3^{6m}}/(\mathbb{F}^*_{3^{6m}})^r$$

r は $r \mid \#E(\mathbb{F}_{3^m})$ となるような大きな素数であり, $r \mid (3^{6m} - 1)$ を満たすことが知られている．$E(\mathbb{F}_{3^m})$ の元は, 点 $Q = (x, y) \in E(\mathbb{F}_{3^m})$ を $E(\mathbb{F}_{3^{6m}})$ へリフティングする distortion 写像 $\phi(x, y) = (-x + \rho, y\sigma)$ を利用して生成できる．つまり, η_T ペアリングの入力は $E(\mathbb{F}_{3^m})$ から元を二つ選ぶこととなる．η_T ペアリングは $P, Q \in E(\mathbb{F}_{3^m})[r], a \in \mathbb{Z}$ に対し, 双線形性 $\eta_T(aP, Q) = \eta_T(P, aQ) = \eta_T(P, Q)^a$ を満たす．この η_T ペアリングと Tate ペアリングの出力の関係は以下のようになっている．

$$\eta_T(P, Q)^{3(3^{(m+1)/2}+1)^2} = e(P, Q)^{-3^{(m+3)/2}}$$

ペアリング暗号の攻撃には, 楕円曲線 $E(\mathbb{F}_{3^m})$ または有限体 $\mathbb{F}_{3^{6m}}$ の離散対数問題を解く方法がある．楕円曲線の離散対数問題を解くアルゴリズムとしては, r のビット長に対する指数関数時間 $\mathcal{O}(\sqrt{r})$ のアルゴリズムが知られており, r は 160 ビット以上であることが必要である．一方, 有限体の離散対数問題を解く方法として, 有限体位数のビット長に対する準指数関数時間アルゴリズムが知られている．標数 p(大きな素数) の有限体に対しては一般数体篩法があり, 標数 2,3 など小さな標数の場合は関数体篩法が適応できる．ここで, 漸近的な計算量として関数体篩法は一般数体篩法より高速と見積もられている[7)]．関数体篩法の大規模実験の結果などを踏まえて, 小さな標数の有限体のサイズをより大きく選択する必要がある．そのため, 本稿では拡大体のサイズ 3^{6m} が 1024 ビット以上の拡大次数 $m = 167, 193, 239$ [⋆1]に対しても η_T ペアリングの実装を行った．

3.2 η_T ペアリングのアルゴリズム

本節では η_T ペアリングのアルゴリズムについて説明する．η_T ペアリングはメインループ部と最終冪の計算からなる．η_T ペアリングは Duursma-Lee アルゴリズム[5)] のループ回数を約半分に減少させたアルゴリズムである[1)]．

メインループは Beuchat らによって Barreto らの η_T ペアリングのアルゴリズムから 3 乗根の計算を除去することにより高速化したアルゴリズムが示された[3)]．最終冪は, メインループの結果 $f \in \mathbb{F}_{3^{6m}}$ と整数 $s = (3^{3m} - 1)\,(3^{3m} + 1)(3^{3m} - 3^{(m+1)/2} + 1)$ に対して f^s を求めるステップである．従来は 3 乗算を繰り返すことで冪乗を計算していたが, 白勢らによってトーラス T_2 を用いることで高速化されたアルゴリズムが示された[20)]．η_T ペアリングの改良されたアルゴリズムとして, Beuchat らによって従来のループの 2 つを 1 つにまとめ, ループの回数を半分にし高速化したアルゴリズムが示された[4)]．改良 η_T ペアリングのアルゴリズムを Algorithm 1 に示す．

3.3 基礎体 $\mathbb{F}_3$, 有限体 $\mathbb{F}_{3^m}$ の元の表現

素体 $\mathbb{F}_3 \in \{0, 1, 2\}$ は状態が 3 つあり, 1 ビットでは表すことができない．そこで hi ビット, lo ビットの 2 ビットで $\mathbb{F}_3$ の元を表現する．$a \in \mathbb{F}_3$ として a の hi ビットを a_{hi}, a の lo ビットを a_{lo} と表現する．

有限体の元 $A(x) \in \mathbb{F}_{3^m}$ は, 多項式表現では

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_0x^0 \tag{1}$$

と表すことができる．ここで $a_i (i = 0, 1, 2, \cdots, m-1)$ は基礎体 $\mathbb{F}_3$ の元である．有限体の元 $A(x) \in \mathbb{F}_{3^{97}}$ をあらわすには 97 個の hi ビット, lo ビットが必要になるが, 97 ビットの型というのは存在しない．そこで a_{hi} と a_{lo} の二つの配列を使って, 値を保持する．ATmega128L のワード長 W は 8 ビットであるため, $\lceil 97/8 \rceil = 13$ となり, 13 個の配列が二つ必要になる．配列の j 番目の要素は $A(x)$ の (hi, lo) をまとめて $A[j]$ と表す．つまり, $W = 8$, $m = 97$ の場合は $A[12] = (0, 0, 0, \cdots, 0, a_{96}), A[11] = (a_{95}, a_{94}, \cdots, a_{88}), \cdots, A[0] = (a_7, a_6, \cdots, a_0)$

⋆1 $m = 97, 167, 193, 239$ は文献 8) が推奨した拡大次数である．

Algorithm 1　改良 η_T ペアリング[4)]

INPUT：　$P=(x_p,y_p), Q=(x_q,y_q)\in E(\mathbb{F}_{3^m})[r]$
OUTPUT：　$\eta_T(P,Q)\in \mathbb{F}^*_{3^{6m}}$
1：　$y_p \leftarrow -y_p,\ \ d\leftarrow 1$
2：　$f\leftarrow -y_p(x_p+x_q+1)+y_q\sigma+y_p\rho$
3：　$u\leftarrow x_p+x_q+d$
4：　$g\leftarrow y_qy_p\sigma-u^2-u\rho-\rho^2$
5：　$f\leftarrow fg$　　$(Algorithm\ 2)$
6：　$y_p\leftarrow -y_p, x_q\leftarrow x_q^9, y_q\leftarrow y_q^9$
7：　$d\leftarrow d-1,\ f\leftarrow f^3$
8：　**for**　$i\leftarrow 0$ to $(m-1)/4-1$
9：　　$u\leftarrow x_p+x_q+d$
10：　　$g_1\leftarrow (y_qy_p\sigma-u^2-u\rho-\rho^2)^3$
11：　　$y_p\leftarrow -y_p, x_q\leftarrow x_q^9, y_q\leftarrow y_q^9$
12：　　$u\leftarrow x_p+x_q+d-1$
13：　　$g_2\leftarrow y_qy_p\sigma-u^2-u\rho-\rho^2$
14：　　$y_p\leftarrow -y_p, x_q\leftarrow x_q^9, y_q\leftarrow y_q^9$
15：　　$d\leftarrow d+1$
16：　　$g\leftarrow g_1g_2$　　$(Algorithm\ 2)$
17：　　$f\leftarrow (f^3g)^3$
18：　**end for**
19：　**return**　f

となる．また, 本稿で使用する記号を以下に定義する．

N:有限体を格納する配列の要素数=$\lceil m/W\rceil$
A & B：論理積
$A\gg k$ ：k ビット右シフト
$A\ll k$ ：k ビット左シフト
$A[j]_k$：配列 A の j 番目の要素の k ビット目

3.4　有限体 $\mathbb{F}_{3^m}$ の演算

$A(x), B(x)\in\mathbb{F}_{3^m}$ に対する加算・減算は, 各係数の (hi, lo) に対し, 論理演算 AND,OR, XOR を用いて演算を行った[8)]．論理演算はそれぞれ 7 回使っている．有限体の乗算は, 改良 Comb 法を用いた．有限体の乗算, 3 乗算については 4 章で詳しく説明する．また, $A(x)\in\mathbb{F}_{3^m}$ に対する乗法逆元は標数 2 のユークリッド互除法を改良した拡張ユークリッド互除法を用いて実装した[11)]．乗法逆元はペアリング全体として 1 回しか使用しないため, ATmega128L に特化した特別な最適化は行っていない．

3.5　拡大体 $\mathbb{F}_{3^{3m}}$, $\mathbb{F}_{3^{6m}}$ の演算

本稿では $\mathbb{F}_{3^m}$ の 6 次拡大体 $\mathbb{F}_{3^{6m}}$ は 3 次拡大した後に 2 次拡大を行うことにより構成する．3 次拡大体の多項式表現は $a_0,a_1,a_2\in\mathbb{F}_{3^m}$ とすると, $a_2\rho^2+a_1\rho+a_0$ となる．3 次拡大の既約多項式は $\rho^3-\rho-1$ とした．また, 2 次拡大体の多項式表現は $\alpha_0,\alpha_1\in\mathbb{F}_{3^{3m}}$ とすると, $\alpha_1\sigma+\alpha_0$ となる．2 次拡大の既約多項式は σ^2+1 とした．6 次拡大体 $\mathbb{F}_{3^{6m}}$ の元 A は $\alpha_j\in\mathbb{F}_{3^{3m}}$, $j=\{0,1\}$, $a_i\in\mathbb{F}_{3^m}$, $i=0,1,\cdots,5$ とすると,

$$\begin{aligned}A&=\alpha_1\sigma+\alpha_0\\&=a_5\sigma\rho^2+a_4\rho^2+a_3\sigma\rho+a_2\rho+a_1\sigma+a_0\\&=(a_5,a_4,a_3,a_2,a_1,a_0)\end{aligned}$$

と表す．$\mathbb{F}_{3^{3m}}$, $\mathbb{F}_{3^{6m}}$ の加算, 減算, 3 乗算, 乗法逆元の演算は文献 8) と同様の演算を行っている．6 次拡大体の乗算は, 文献 8) では $\mathbb{F}_{3^m}$ の乗算回数が 18 回必要となるが, Gorla らによって 15 回で行うことができる方法が示された[6)]．表 2 に各演算に必要な $\mathbb{F}_{3^m}$ の演算回数を示す．A は加算, M は乗算, C は 3 乗算, I は乗法逆元である．

Algorithm 1 のステップ 5,16 で現れる, 定数項を含む 6 次拡大体の乗算として

$$f,g\in\mathbb{F}_{3^{6m}}, f=(0,a,0,f_2,f_1,f_0), g=(0,-1,0,g_2,g_1,g_0), a\in\{0,-1\}$$

となる場合がある．この乗算は 6M で計算することが可能である．このアルゴリズムを Algorithm 2 に示す．

η_T ペアリングに必要な演算コストを表 3 に示す．Barreto らの η_T ペアリング[1)] と, Algorithm 1 と Gorla らの高速 6 次拡大体乗算[6)] を適用した η_T ペアリングを比較すると, およそ 130 回乗算が少ないことがわかる．一方, 3 乗算の計算回数が約 70 回増加しているが, 3 乗算のコストは乗算のコストの 1/10 以下であるため, 乗算 7 回程度の計算量である．そのため, 改良 η_T ペアリング (Algorithm 1) の方が高速となる．

表 2　$\mathbb{F}_{3^{3m}}, \mathbb{F}_{3^{6m}}$ の演算に必要な $\mathbb{F}_{3^m}$ の演算回数
Table 2　Number of times of addition in $\mathbb{F}_{3^m}$ for operations in $\mathbb{F}_{3^{3m}}, \mathbb{F}_{3^{6m}}$

演算	$\mathbb{F}_{3^{3m}}$	$\mathbb{F}_{3^{6m}}$
加減算	3A	6A
乗算	12A + 6M	51A + 15M
3 乗算	3A + 3C	6A + 6C
乗法逆元	6A + 15M + 1I	57A + 38M + 1I

表 3　η_T ペアリングに必要な $\mathbb{F}_{3^m}$ の演算回数
Table 3　Number of times of operations in $\mathbb{F}_{3^m}$ for η_T pairing

従来の η_T ペアリング	784C + 820M + I
改良 η_T ペアリング (Alg. 1)	852C + 693M + I

Algorithm 2　定数項を含む $f, g \in \mathbb{F}_{3^{6m}}$ の乗算

INPUT :　$f = (0, a, 0, f_2, f_1, f_0),\ g = (0, -1, 0, g_2, g_1, g_0) \in \mathbb{F}_{3^{6m}}, a = \{-1, 0\}$
OUTPUT :　$c = f \cdot g = (c_5, c_4, c_3, c_2, c_1, c_0) \in \mathbb{F}_{3^{6m}}$
1 :　$m_0 \leftarrow f_0 g_0,\ m_1 \leftarrow f_1 g_1,\ m_2 \leftarrow f_2 g_2$
2 :　$m_3 \leftarrow (f_0 + f_1)(g_0 + g_1)$
3 :　$m_4 \leftarrow (f_0 + f_2)(g_0 + g_2)$
4 :　$m_5 \leftarrow (f_0 + f_1 + f_2)(g_0 + g_1 + g_2)$
5 :　$c_0 \leftarrow m_0 - m_1 - g_2$
6 :　$c_1 \leftarrow m_3 - m_0 - m_1$
7 :　$c_2 \leftarrow m_4 - m_0 - m_2 - f_2 - g_2$
8 :　$c_3 \leftarrow m_5 - m_3 - m_4 + m_0$
9 :　$c_4 \leftarrow m_2 - g_0 - f_0$
10 :　$c_5 \leftarrow g_1$
11 :　**if**　$a = -1$
12 :　　$c_0 \leftarrow c_0 - f_2,\ \ c_2 \leftarrow c_2 + 1$
13 :　　$c_4 \leftarrow c_4 + 1,\ \ c_5 \leftarrow c_5 + f_1$
14 :　**end if**
15 :　**return**　$c = (c_5, c_4, c_3, c_2, c_1, c_0)$

4. ATmega128L に特化した実装方法

本節では, 3 節で説明した η_T ペアリングを, ATmega128L に特化して高速化するために行った実装方法について説明する.

4.1 $\mathbb{F}_{3^m}$ の乗算

有限体の乗算は多項式乗算とリダクションからなる. 多項式乗算として Shift Add 法がある[10]. Shift Add 法は $A(x)$ を 1 回ずつ左シフトを行い, $B(x)$ を 1 項ずつ走査し加算を行う方法である. そのためシフトと加算がそれぞれ $m-1$ 回必要となる.

多項式乗算を行うもう一つのアルゴリズムとして Comb 法がある[10]. Comb 法はワード長おきに次数 m まで走査し, その後でシフトを行う方法である. さらに, 吉冨らにより改良 Comb が提案された[23]. 改良 Comb 法は, $W \nmid m$ となる場合, m 次以上の項を走査しないように改良されたアルゴリズムである. 改良 Comb 法では, シフトの回数が必ず $W-1$ 回となる. そのためワード長が小さい環境ではシフトの回数を大幅に削減することが可能である.

ATmega128L はワード長が 8 ビットと小さいため, 高々7 回のシフトで計算することができる. 実装の結果, $\mathbb{F}_{3^m}$ の乗算に要する時間は, Shift Add 法では $14m$sec, 改良 Comb 法では $6.2m$sec となった. このアルゴリズムを Algorithm 3 に示す.

4.2 $\mathbb{F}_{3^m}$ の 3 乗算

$A(x) \in \mathbb{F}_{3^m}$ に対する 3 乗算の高速アルゴリズムについて説明する. 初期実装では, 以下の 2 つの処理に分けて実装を行った.

- $A(x)^3 = \sum_{i=0}^{m-1} a_i x^{3i}$ を求める多項式 3 乗算
  $A(x) \in \mathbb{F}_{3^m}$ に対する多項式 3 乗算は $A(x)^3 = \sum_{i=0}^{m-1} a_i x^{3i}$ となる. この性質を使い, 各係数を引き伸ばすテーブルを作成する. そしてこのテーブルに従って各係数を引き伸ばす処理を行うことによって多項式 3 乗算を高速に行う. ATmega128L はワード長が 8 ビットであるため, 3 ビット, 3 ビット, 2 ビットの順で引き伸ばしテーブルを利用する. 引き伸ばしテーブルの構成を図 1 に示す. 引き伸ばしテーブルのサイズは 8 バイト (2^3) となる.
- $C(x) = A(x)^3 \bmod f(x)$ を求めるリダクション

この方法はペアリングの代表的な実装方法として用いられている[11),23]. しかし, 図 1 に示したように, このテーブルでは一度に引き伸ばせるのは 3 ビットまでである. また, 引き伸ばした結果, 次数が約 3 倍になるため, リダクション処理において必要な有限体の加減算の回数が乗算のリダクションより多くなってしまう. この引き伸ばし処理を用いた 3 乗算の

Algorithm 3　Refined Comb Method[23] + Reduction with ROTs[15]

INPUT : $A(x)=\sum_{i=0}^{m-1}a_ix^i, B(x)=\sum_{i=0}^{m-1}b_ix^i, f(x)=x^m+x^k+2, W|k$
OUTPUT : $C(x)=A(x)\cdot B(x) \bmod f(x)=\sum_{i=0}^{m-1}c_ix^i \in \mathbb{F}_{3^m}$

1 : $C \leftarrow 0$
2 : **for** $j \leftarrow 0$ to $N-1$
3 : $\quad C(x) \leftarrow C(x)+A[j]\cdot B(x)x^{jW}$
4 : **end for**
5 : **for** $i \leftarrow 1$ to $W-1$ **do**
6 : $\quad$ **for** $j \leftarrow 0$ to $N-2$ **do**
7 : $\quad\quad C(x) \leftarrow C(x)+A[j]_i\cdot B(x)x^{jW+i}$
8 : $\quad$ **end for**
9 : **end for**
10 : **for** $i \leftarrow 2m-2$ to m **do**
11 : $\quad c_{i-m+k} \leftarrow c_{i-m+k}-c_i$
12 : $\quad c_{i-m} \leftarrow c_{i-m}+c_i$
13 : $\quad c_i \leftarrow 0$
14 : **end for**
15 : **return** $C(x)$

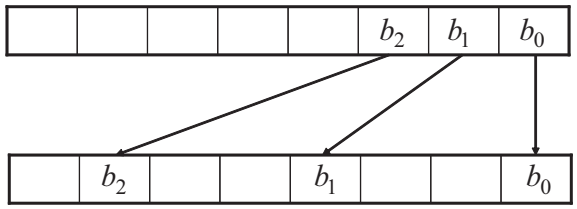


図 1　$\mathbb{F}_{3^m}$ の 3 乗算で用いる引き伸ばしテーブル

アルゴリズムを Algorithm 4 に示す.

本稿では引き伸ばし処理ではなく並び替え処理を用いることによって高速化したアルゴリズムを提案する. この方法ではまず, 多項式 3 乗算とリダクションを行った結果を事

Algorithm 4　引き伸ばし処理を用いた $\mathbb{F}_{3^m}$ の 3 乗算

INPUT : $A(x)=\sum_{i=0}^{m-1}a_ix^i \in \mathbb{F}_{3^m},\ f(x)=x^m+x^k+2$
OUTPUT : $C(x)=A(x)^3 \bmod f(x)=\sum_{i=0}^{m-1}c_ix^i \in \mathbb{F}_{3^m}$

1 : **for** $i=0$ to $3m-3$
2 : $\quad$ **if** $3 \not| i$ **then**
3 : $\quad\quad c_i \leftarrow 0$
4 : $\quad$ **else**
5 : $\quad\quad c_{3i} \leftarrow a_i$
6 : $\quad$ **end if**
7 : **end for**
8 : **for** $i \leftarrow 3m-3$ to m **do**
9 : $\quad c_{i-m+k} \leftarrow c_{i-m+k}-c_i$
10 : $\quad c_{i-m} \leftarrow c_{i-m}+c_i$
11 : $\quad c_i \leftarrow 0$
12 : **end for**
13 : **return** $C(x)$

前に計算しておく必要がある. 例えば, $m=97, A(x)=(a_{96},a_{95},\cdots,a_1,a_0), B(x)=(b_{96},b_{95},\cdots,b_1,b_0) \in \mathbb{F}_{3^m}, f(x)=x^{97}+x^{16}+2$ の場合,

$$B(x)=A(x)^3 \bmod f(x)$$

の計算を行うと, 各係数は以下のようになる.

$b_0=a_0$　　$b_8=a_{35}+a_{89}-a_{62}$　　$b_{96}=a_{32}+a_{86}-a_{59}$
$b_1=a_{65}+a_{92}$　　$b_9=a_3$
$b_2=a_{33}+a_{87}-a_{60}$　　$b_{10}=a_{68}+a_{95}$
$b_3=a_1$　　$b_{11}=a_{36}+a_{90}-a_{63}$　　$\cdots$
$b_4=a_{66}+a_{93}$　　$b_{12}=a_4$
$b_5=a_{34}+a_{88}-a_{61}$　　$b_{13}=a_{69}+a_{96}$
$b_6=a_2$　　$b_{14}=a_{37}+a_{91}-a_{64}$
$b_7=a_{67}+a_{94}$　　$b_{15}=a_5$

この結果から, $0<k<11$ の場合,

$$\underbrace{(b_{8k+5}, b_{8k+2})}_{\text{第三系列}},\ \underbrace{(b_{8k+7}, b_{8k+4}, b_{8k+1})}_{\text{第二系列}},\ \underbrace{(b_{8k+6}, b_{8k+3}, b_{8k})}_{\text{第一系列}}$$

の組み合わせは, a の係数が連続していることがわかる. それぞれ第一系列, 第二系列, 第三系列と呼ぶことにする.

3 節で説明したように有限体の加減算は, ワード長単位で行うことが出来るため, 第一系列の 3 ビット, 第二系列の 3 ビット, 第三系列の 2 ビットの加減算を一度に計算することが出来る. そのために, 8 ビットの変数に図 2 のように値を入れる. 例えば $b_0 \sim b_7$ を計算する場合, 第一系列は $(b_0, b_3, b_6) = (a_0, a_1, a_2)$ となる. 第二系列は 2 項の和になっているので, $(b_1, b_4, b_7) = (a_{65}, a_{66}, a_{67}) + (a_{92}, a_{93}, a_{94})$ となる. 同様に第三系列は $(b_2, b_5) = (a_{33}, a_{34}) + (a_{87}, a_{88}) - (a_{60}, a_{61})$ と表せる.

最も有限体の加減算が必要なのは第三系列の 2 回である. そのため, $b_0 \sim b_7$ は図 2 のように 3 つの変数に値を入れ, 有限体の加減算 2 回を行うと $b_0 \sim b_7$ までが計算できることになる.

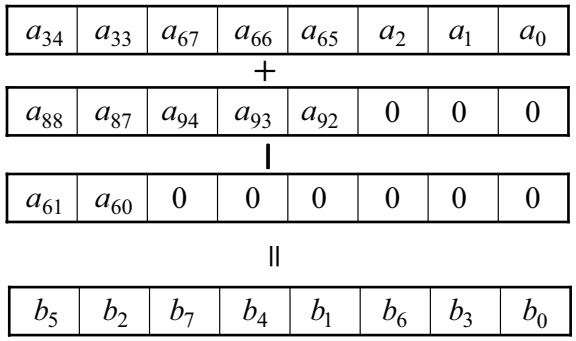


図 2　$b_0 \sim b_7$ の計算

その後, 各係数を並び替える処理を行う. この処理は図 3 のように値を入れ替える処理である. この処理には事前計算した並び替えテーブルを使用する. 並び替えテーブルのサイズは 256 バイト (2^8) 必要になる.

初期実装では 3 ビットごとに引き伸ばしの処理を行ったが, 並び替え処理では 8 ビットごとに処理を行うことが出来るため高速に計算を行うことが出来る. この処理にかかるコストは, シフトとマスク処理, 有限体の加減算 2 回である. また, 引き伸ばし処理に伴うリダクションの処理も必要ない.

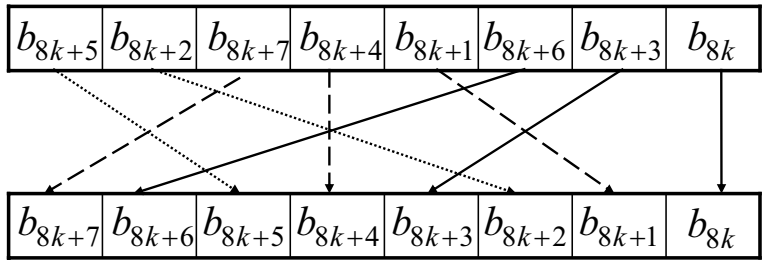


図 3　並び替え処理 (Table)

Algorithm 5　並び替え処理を用いた $\mathbb{F}_{3^m}$ の 3 乗算

INPUT:　$A(x) = \sum_{i=0}^{m-1} a_i x^i \in \mathbb{F}_{3^m},\ f(x) = x^m + x^k + 2$
OUTPUT:　$C(x) = A(x)^3 \bmod f(x) = \sum_{i=0}^{m-1} c_i x^i \in \mathbb{F}_{3^m}$

```
1 :  C(x) ← 0
2 :  for i ← 0 to ⌈m/W⌉
3 :    for j ← 0 to N(i) − 1
4 :      C[i] ← C[i] + (F(1,i) & F(2,i) & F(3,i))
5 :    end for
6 :    C[i] ← Table(C[i])
7 :  end for
8 :  return C
```

ただし, この計算方法が可能であるのは $f(x) = x^{97} + x^{16} + 2$ のように既約多項式の中間の項の次数がワード長の整数倍の場合である. 他の既約多項式の場合は配列のそれぞれの要素において異なる並び替えテーブルが必要となるので効率的ではない. このテーブルを利用した方法は, 引き伸ばし処理を行う方法に比べ, 80%高速化することが出来た. 並び替え処理を用いた $\mathbb{F}_{3^m}$ の 3 乗算を Algorithm 5 に示す. ステップ 4 の $F(s, j)$, $s = 0, 1, 2, j = 0, 1, \cdots, \lceil m/W \rceil$ はそれぞれ入力された要素番号 j に対応した第 s 系列の値を返す関数である. この値は複数存在し, 任意に順序付ける必要がある. たとえば $m = 97$ の $F(0, 0)$ は $a_0 + a_1 * 2 + a_3 * 4$

表 4　$\mathbb{F}_{3^m}$ の 3 乗算に必要なワード長ごとの有限体上の加算回数
Table 4　Number of times of addition for cubing in $\mathbb{F}_{3^m}$

		加算回数			
次数 m	既約多項式	Alg. 4	提案方式 Alg. 5	速度 ($m$sec)	選択
97	$x^{97}+x^{16}+2$	100	30	0.40	○
167	$x^{167}+x^{96}+2$	172	36	0.69	○
193	$x^{193}+x^{64}+2$	196	66	1.15	○
	$x^{193}+x^{112}+2$	196	102		
239	$x^{239}+x^{24}+2$	236	57		
	$x^{239}+x^{56}+2$	236	74		
	$x^{239}+x^{96}+2$	236	48	1.16	○
	$x^{239}+x^{104}+2$	236	95		

となる．ステップ 3 の，$N(i)$ は，i 番目の要素の第一〜第三系列の中の最大となる個数である．$m=97$ の場合は $N(0)=3, N(1)=3, \cdots, N(12)=1$ となる．したがってワード長ごとの有限体上の加算回数は $\sum_{j=0}^{\lceil m/W \rceil}\{N(j)-1\}$ となる．また，実装上では $F(s,j), N(i)$ は事前計算した値を用いる．

4.3　既約多項式の選択

既約多項式を適切に選択することによって有限体のリダクション，3 乗算を高速化することができる．

4.3.1　有限体 $\mathbb{F}_{3^m}$ のリダクションの高速化

有限体 $\mathbb{F}_{3^m}$ の既約多項式を $f(x)=x^m+ax^k+b, (a,b)=(1,2)$ または $(2,1), 0<k<m$ とすると，中間項の次数 k が最小になるような既約多項式が選択されてきた[11]．しかし，Nakajima らによってリダクションに特化した既約多項式 (Reduction Optimal Trinomials, ROTs) を利用することによってリダクション処理を高速に計算することが可能となることが示された[15]．ROTs は k を $W|k$ を満たすように選択する．ATmega128L はワード長 W は 8 ビットとなるので，$8|k$ となる既約多項式を選択する．

表 4 に次数 97, 167, 193, 239 についての ROTs を示した．

次数 $m=97$ の場合，$f(x)=x^{97}+x^{12}+2$ を利用したリダクションよりも $f(x)=x^{97}+x^{16}+2$ を利用したリダクションの方が，約 38%高速である．$m=97, W=8$ の場合のリダクションのアルゴリズムを Algorithm 3 に示した．

4.3.2　並び替え処理を用いた 3 乗算の高速化

4.2 節で説明した並び替え処理を用いた 3 乗算の計算量の殆どは，有限体 $\mathbb{F}_{3^m}$ の加算である．有限体 $\mathbb{F}_{3^m}$ の 3 乗算で使う加算の回数は次数 m と既約多項式 $f(x)$ によって決定される．そのため，既約多項式による演算回数の比較を検討する必要がある．配列の 2 要素に対して有限体上の加算を行い，有限体の元 8 個ごと，つまりワード長ごとに論理演算 7 回を用いて並列計算を行うことができる．各係数の事前計算を行った結果から，ワード長ごとの有限体上の加算の回数を算出した．

また，引き伸ばし処理を用いた場合の加算回数をあわせて比較した．各次数，既約多項式による加算回数の違いを表 4 に示した．また選択した既約多項式における $\mathbb{F}_{3^m}$ の 3 乗算の ATmega128L での実装結果を示した．

表から既約多項式の選択を適切に行うことにより，3 乗算も高速化することが可能であることがわかる．$m=97, 167$ の場合は ROTs が一つしか存在しないが，$m=193, 239$ の場合は有限体 $\mathbb{F}_{3^m}$ の加算回数が最も少なくなるように選択することにより高速化することが出来る．特に，次数 $m=239$ の場合は，最も回数が多い $f(x)=x^{239}+x^{104}+2$ と最も少ない $f(x)=x^{239}+x^{96}+2$ を比較すると，有限体 $\mathbb{F}_{3^m}$ の加算回数が約半分になることがわかる．

4.4　速 度 評 価

速度評価は有限体 $\mathbb{F}_{3^m}$ の演算，η_T ペアリングの処理速度について行う．MICAz では TinyOS 上に NesC 言語で実装した．本実装で用いる MICAz のスペックは，2 節で述べた従来技術の TinyTate でのスペックと同じである (表 1 を参照)．

NesC は C 言語を拡張した言語であり，関数は C 言語とほぼ同じ記述をすることが出来る．

3 節で示した有限体の各種演算，従来の公開鍵暗号である η_T ペアリングのアルゴリズムでの初期実装は約 30sec であった．そして本稿で説明した既約多項式の選択，有限体乗算，並び替えテーブルを用いた 3 乗算の高速化の結果，約 8sec まで高速化することができた．加えて Gorla らの 6 次拡大体乗算の高速化[6]，Beuchat らの改良 η_T ペアリング[4] を適用した結果，約 5.8sec まで高速化された．

表 5 に拡大次数における η_T ペアリングの実行時間の比較を示した．PC は Core 2 Duo E6400, 1.86GHz, メモリ 1GB, FedoraCore6 を用い C 言語で実装した．コンパイラは GCC を使用し，-O3 -fomit-frame-pointer -unroll-loops で最適化を行った．

使用した PC は 32 ビット CPU であるが，ATmega128L は 8 ビットであるため PC においても $W=8$ ビット (unsigned char) で実装を行った．PC と ATmega128L での同一のコードの速度を比較するため，表 6 に PC での実装データを載せた．ATmega128L で使用したペアリング関数と PC の実装で用いたペアリング関数は同じものを使用している．ATmega128L では PC のおよそ 1,000 倍の計算時間がかかっていることがわかる．なお，表 5 及び 6 では，ATmega128L による有限体演算，及び η_T ペアリングの計算時間に対しては

表 5　有限体 $\mathbb{F}_{3^m}$ 上の演算と η_T ペアリングの計算時間
Table 5　Timing of arithmetic of $\mathbb{F}_{3^m}$ and the η_T pairing

演算	計算時間 ($m$sec)			
	$\mathbb{F}_{3^{97}}$	$\mathbb{F}_{3^{167}}$	$\mathbb{F}_{3^{193}}$	$\mathbb{F}_{3^{239}}$
加算	0.047	0.076	0.084	0.092
減算	0.048	0.078	0.084	0.092
乗算	6.2	18.2	25.5	35.75
3 乗算	0.40	0.69	1.15	1.16
逆元算	96	251.5	360.8	480.7
η_T ペアリング	5.8×10^3	15.3×10^3	34.6×10^3	60.2×10^3

CPU: ATmega128L，クロック周波数: 7.37MHz，ワード長: 8 ビット

表 6　拡大次数 m による η_T ペアリングの計算時間の比較
Table 6　Comparison of timing of the η_T pairing on several extension fields

次数 m	ATmega128L ($m$sec)	PC(W=8)($m$sec)
97	5.8×10^3	6.99
167	15.3×10^3	25.74
193	34.6×10^3	41.96
239	60.2×10^3	65.61

PC：Core 2 Duo，クロック周波数：1.86GHz
(注意：ATmege128L と同一のコードを使用)

それぞれ 1,000, 及び 1,000,000 回の実測結果の平均値を表記しており, PC による η_T ペアリングの計算時間には, 10,000 回の実測結果の平均値を表記している.

TinyOS 上への実装結果の比較を表 7 に示した. TinyPK は 1024 ビットの公開鍵を法とした冪乗算 (冪は $e = 3$) の実行時間, TinyECC は 160 ビットの標数 p の楕円曲線のスカラー倍算の時間である. また, これらの実行時間には入出力に要する時間は含まれていない. インラインアセンブリを用いた TinyECC が最も高速高速な結果となっている. Tate ペアリングである TinyTate[16] と比較すると, η_T ペアリングを用いた本実装では約 5 倍程度高速であるという結果となった. TinyPBC と比較すると, 実行速度は TinyPBC の方が高速であり, RAM の使用量も少ない. ROM は本実装の方が TinyPBC の約 3 分の 1 となっている. 本実装においての RAM 使用量の内訳としては, 並び替えテーブルに 256 バイト使用し, グローバル変数として残りの RAM を使っている. 特にカウンタ変数やバッファ用の変数を様々な関数で何度も宣言するコストを削減するために, グローバル変数として使った.

表 7　TinyOS を用いた ATmega128L 上での公開鍵暗号実装の比較
Table 7　Comparison of timing of public key cryptosystems on ATmega128L using TinyOS

	TinyPK[24]	TinyECC[12]	TinyTate[16]	TinyPBC[17]	本実装
言語	NesC	NesC,asm	NesC	NesC	NesC
暗号方式	RSA	ECC	Tate	η_T(標数 2)	η_T(標数 3)
ROM	12,408	13,858	18,384	47,948	17,284
RAM	1,167	1,440	1,831	368	628
実行時間	14.5	1.9	30.2	5.5	5.8

ROM,RAM: bytes，実行時間: sec，クロック周波数: 7.37MHz，ワード長: 8 ビット

5. おわりに

本稿では ATmega128L に標数 3 の体 $\mathbb{F}_{3^m}$ 上での η_T ペアリングの実装を行った. 実装は TinyOS 用いて MICAz 上に行った. また, ワード長を 8 ビットとして PC 上での実装も行い, ATmega128L との比較を行った. ペアリング暗号は有限体 $\mathbb{F}_{3^m}$, 拡大体 $\mathbb{F}_{3^{3m}}, \mathbb{F}_{3^{6m}}$ の演算を使用する. そのため, これらの演算の高速化もあわせて行った.

ATmega128L 初期実装では約 30sec 計算時間がかかっていた. 特に, 有限体 $\mathbb{F}_{3^m}$ の乗算と 3 乗算の実行時間が支配的だったため, それらの演算を重点的に高速実装を行った. 高速化にあたり ATmega128L に特化した有限体の乗算, 3 乗算の高速化, 有限体の演算回数の削減を行った.

有限体乗算は多項式乗算とリダクションに分かれる. 多項式乗算のアルゴリズムとしては Shift Add 法や Comb 法, Window 法などがある. これらを実装し, その中で一番高速であった改良 Comb 法を選択した[23]. また, 並び替えテーブルを用いた有限体 $\mathbb{F}_{3^m}$ 上の 3 乗算を提案し, 約 5 倍の高速化を達成した. さらに, 中間項の次数がワード長 W の整数倍となる既約多項式 (ROTs[15]) : $f(x) = x^m + x^k + 2, W|k)$ を選択することにより, リダクション, 3 乗算を高速化した.

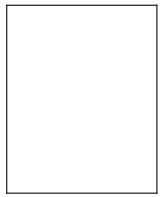
上記の有限体の演算の高速化を行うことによって約 8sec まで高速化された. また, 有限体の演算を削減するために, Gorla らによって示された 6 次拡大体の乗算, Beuchat らによって示された η_T ペアリングのメインループのアルゴリズムを適用した. その結果, η_T ペアリングの計算時間を約 5.8sec に改善することができた.

TinyOS 上での Tate ペアリングである TinyTate に比べて約 6 倍の高速化を達成した. これにより, TinyOS を利用した ATmega128L 上でのペアリング暗号は, 従来の RSA, 楕円曲線暗号と同程度の速度で実装することが可能となった.

参 考 文 献

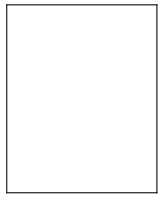
1) Barreto, P., Galbraith, S.,Ó hÉigeartaigh, C. and Scott, M. : Efficient Pairing Computation on Supersingular Abelian Varieties, Designs, Codes and Cryptography, pp.239-271 (2004).
2) Boneh, D. ,Gentry, C. and Waters, B. : Collusion Resistant Broadcast Encryption With Short Ciphertexts and Private Keys, *CRYPTO2005*, LNCS 3621, pp.258-275 (2005).
3) Beuchat, J.-L., Shirase, M., Takagi, T. and Okamoto, E. : An algorithm for the η_T pairing calculation in characteristic three and its hardware implementation, *Proc. 18th IEEE International Symposium on Computer Arithmetic, ARITH-18*, pp.97-104 (2007).
4) Beuchat, J.-L., Shirase, M., Takagi, T. and Okamoto, E. : A Refined Algorithm for the η_T Pairing Calculation in Characteristic Three, Cryptology ePrint Archive, Report 2007/311 (2007).
5) Duursma, I. and Lee, H. :Tate pairing implementation for hyperelliptic curve $y^2 = xp - x + d$, *ASIACRYPT2003*, LNCS 2894, pp.111-123 (2003).
6) Gorla, E., Puttmann, C. and Shokrollahi, J. : Explicit Formulas for Efficient Multiplication in $\mathbb{F}_{3^{6m}}$, *SAC2007*, LNCS 4876, pp.173-183 (2007).
7) Granger, R., Holt, A., Page, D., Smart, N. and Vercauteren, F. : Function Field Sieve in Characteristic Three, *ANTS 2004*, LNCS 3076, pp.223-234 (2004).
8) Granger, R., Page, D. and Stam, M. : On Small Characteristic Algebraic Tori in Pairing-Based Cryptography, *LMS Journal of Computation and Mathematics, Vol.9, pp.64-85 (2006).*
9) *Gura, N., Patel, A., Wander, A., Eberle, H. and Shantz, S.C. : Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs, CHES2004, LNCS 3156, pp.119-132 (2004).*
10) *Hankerson, D., Menezes, A. and Vanstone, S. : Guide to Elliptic Curve Cryptography, Springer, pp.48-49 (2004).*
11) 川原祐人, 高木剛, 岡本栄司 *:Java* を利用した携帯電話上での *Tate* ペアリングの高速実装, 情報処理学会論文誌, *Vol.49, No.1, pp.427-435 (2008).*
12) *Liu, A. and Ning, P. :TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks, To appear in Proc. 7th International Conference on Information Processing in Sensor Networks (IPSN2008), SPOTS Track (2008).*
13) *MICAz Hardware Description Available at : http://www.xbow.jp/.*
14) *Miller, V. : Short Programs for Functions on Curves, Unpublished Manuscript, (1986).*
15) *Nakajima T., Izu, T. and Takagi, T. : Reduction Optimal Trinomials for Efficient Software Implementation of the η_T Pairing, 2nd International Workshop on Security, IWSEC2007, LNCS 4752, pp.44-57 (2007).*
16) *Oliveira, L., Aranha, D., Morais, E., Daguano, F., López, J. and Dahab, R. : TinyTate: Identity-Based Encryption for Sensor Networks, Sixth IEEE International Symposium on Network Computing and Applications (NCA 2007), pp.318-323 (2007).*
17) *Oliveira, L., Scott, M., Lopez, J. and Dahab, R. : TinyPBC: Pairings for Authenticated Identity-Based Non-Interactive Key Distribution in Sensor Networks, Cryptology ePrint Archive, Report 2007/482 (2007).*
18) *Perrig, A., Stankovic, J. and Wagner, D. : Security in wireless sensor networks, Communications of the ACM, v. 47 n. 6, 2004. Cryptology ePrint Archive, Report 2007/340 (2007).*
19) *Ronan, R., Ó hÉigeartaigh, C., Murphy, C. , Kerins, T. and Barreto, P. : Hardware Implemantation of the η_T pairing in Characteristic 3, Cryptology ePrint Archive Report 2006/371 (2006).*
20) *Shirase, M., Takagi, T. and Okamoto, E. : Some Efficient Algorithms for the Final Exponentiation of η_T Pairing, IEICE Transactions, Vol.E91-A, No.1, pp.221-228 (2008).*
21) 徳田 英幸ら *:* 総務省ユビキタスセンサーネットワーク技術に関する調査研究会「最終報告書」*, pp.591-596 (2006).*
22) *TinyOS : An open-source OS for the networked sensor regime, http://www.tinyos.net/.*
23) *Yoshitomi, M., Takagi, T., Kiyomoto, S. and Tanaka, T. :Efficient Implementation of the Pairing on Mobilephones using BREW, IEICE Transaction, Vol.E91-D, No.5, pp.1330-1337, (2008).*
24) *Watro, R., Kong, D., Cuti, S., Lynn, C. and Knuus, P. :TinyPK:Securing Sensor Networks with Public Key Technology, SASN2004, pp.59-64 (2004).*

(平成？年？月？日受付)
(平成？年？月？日採録)



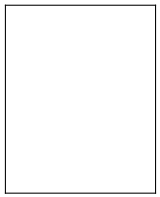
石黒　　司（学生会員）

2008 年公立はこだて未来大学システム情報科学部卒業．現在，情報セキュリティ大学院大学情報セキュリティ研究科博士前期 (修士) 課程在学中．暗号実装および数論アルゴリズムに関する研究に従事．2007 年 CSS2007 学生論文賞．電子情報通信学会会員．



白勢　政明

1994 年茨城大学理学部数学科卒業．1996 年同大学大学院理学研究科修了．同年 JTB 出版入社， JTB 時刻表システムの開発に従事．2006 年北陸先端科学技術大学院大学情報科学研究科博士課程修了，情報科学博士．その後，公立はこだて未来大学ポストドクターを経て，2008 年より同大学システム情報科学部助教，現在に至る．暗号ハードウェア実装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．



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ＮＴＴ情報流通プラットフォーム研究所入社．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 年より同大教授，現在に至る．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．