2007年度 修士論文

$GF(2^n)$及び$GF(P)$における楕円曲線暗号向けスケーラブルモンゴメリ乗算器に関する研究

指導教授　柳澤 政生　教授

早稲田大学大学院 理工学研究科

情報・ネットワーク専攻

3606U120–9

谷村 和幸

Kazuyuki Tanimura

2008年2月4日

目 次

第 1 章　序論 **3**

1.1　本論文の背景と意義 4

1.2　本論文の概要 7

第 2 章　公開鍵暗号 **8**

2.1　本章の概要 9

2.2　暗号の基本 10

2.3　楕円曲線暗号 13

2.4　暗号の安全性 14

2.5　楕円曲線暗号の一般的安全性 15

2.6　本章のまとめ 17

第 3 章　楕円曲線暗号アルゴリズム **18**

3.1　本章の概要 19

3.2　楕円曲線の式 20

3.3　楕円曲線上の加法群 22

3.4　有限体上の演算 25

3.5　座標系 26

3.6　スカラー乗算 28

3.6.1　二進展開法 28

3.6.2　移動窓法 28

3.6.3　符号付表現（NAF） 29

3.6.4　Montgomery 型楕円曲線スカラー乗算 29

3.7　既存手法の比較 33

3.8 本章のまとめ 35

第 4 章 モンゴメリ乗算アルゴリズム 36

4.1 本章の概要 37
4.2 $GF(P)$ におけるモンゴメリ乗算 38
4.3 $GF(2^n)$ におけるモンゴメリ乗算 41
4.4 スケーラブルモンゴメリ乗算 42
4.5 本章のまとめ 44

第 5 章 既存モンゴメリ乗算器 45

5.1 本章の概要 46
5.2 スケーラブル乗算器 47
5.3 ユニファイド型乗算器 49
5.4 本章のまとめ 51

第 6 章 提案モンゴメリ乗算器 52

6.1 本章の概要 53
6.2 提案モンゴメリ乗算器アーキテクチャ 54
6.3 HDL による実装 59
6.3.1 FPGA を用いた実装 59
6.3.2 90nm プロセスライブラリを用いた論理合成結果 62
6.4 性能評価 64
6.4.1 実行時間 64
6.4.2 面積 65
6.5 本章のまとめ 67

第 7 章 結論 68

謝辞 71

参考文献 72

第1章

序論

1.1　本論文の背景と意義

近年，個人情報を含むあらゆる情報のやり取りがコンピュータネットワーク経由で行われている．オンラインショッピングをする際には，クレジットカード番号をインターネット上で送信したり，振込み手続きをオンラインで行うためにキャッシュカード番号やその暗証番号を送信したりする必要がある．そこで，重要になるのが不正改ざん等に対するセキュリティ技術である．

攻撃者は常にインターネットなどのネットワーク上であらゆる手段を講じ，個人情報を手に入れようと試みている．なぜなら，一旦 ID やパスワード等を盗んでしまえば，ネットワーク上で他人に成りすまして取引を行うことが可能だからである．また，近年の携帯電話には個人情報や電子マネーの情報が含まれており，これを紛失した場合，攻撃の対象となりうる．従って，このような不正に対抗するセキュリティ技術の重要性が高まっている．

セキュリティ技術の一つに暗号がある．この暗号は大きく共通鍵暗号と公開鍵暗号の 2 つに分けられる．共通鍵暗号は暗号化したい文章の送信者と受信者で共通の秘密鍵を用いる．送信者と受信者で秘密鍵を共有するためには，一方から他方へ秘密鍵を送信する必要がある．この共通鍵暗号の鍵配送問題を解決するのが公開鍵暗号である．公開鍵暗号では，秘密鍵とは異なる公開鍵を生成し，これを通信したい相手に配送する．このとき，公開鍵は攻撃者に知られても良い．一般的に，公開鍵暗号は共通鍵暗号の秘密鍵を配送するのに用いられる．また公開鍵暗号は，ネットワーク上での認証・署名機能の提供も可能とする．但し，公開鍵暗号は共通鍵暗号より処理時間が長くなるという欠点がある．

一般的に，公開鍵安全性は鍵長により決定される．現在，公開鍵暗号方式のデファクトスタンダードとなっているのは RSA 暗号である．しかし，RSA 暗号の欠点の一つに処理コストが大きいことが挙げられる．具体的には，1024 から 4096 ビット程の剰余乗算，累乗演算を行なわなければならない．従って，リソースの限られた組込みシステム環境 (ワイヤレス携帯端末やスマートカード等) において高い性能を引き出すのは難しい [3]．そこで，注目されているのが楕円曲線暗号である．楕円曲線暗号の鍵長 160 ビットの安全性は，RSA 暗号の鍵長 1024 ビットの安全性と同等とされている [5]．この鍵長が短くても高い安全性を確保できると

いう特長から，楕円曲線暗号のハードウェア実装は，RSA 暗号に比べて小面積化，高速化、低消費電力に向いており，プロセッサの処理能力やメモリの容量，面積，電力が制限されている携帯電話や IC カード等，組み込み用途への採用が期待されている．

楕円曲線暗号の中で，最も支配的な演算は剰余乗算である．剰余乗算では乗算結果を得た後に，除算を実行して剰余を取る必要がある．対して，モンゴメリ乗算 [20] では剰余を取りながら乗算の実行が可能である．そのため，楕円曲線暗号では一般的にモンゴメリ乗算が用いられる．

図 1.1 に楕円曲線暗号の階層図を示す．モンゴメリ乗算は最下層に属するが，これを用いて上位層の計算が行われるため，楕円曲線暗号の高速化にはモンゴメリ乗算器の高速化が最も効果的である [26].

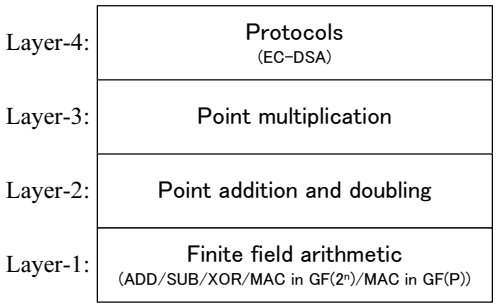

図 1.1: 楕円曲線暗号スキームにおける演算操作の階層.

楕円曲線暗号は素体 ($GF(P)$) もしくは 2 の拡大体 ($GF(2^n)$) 上の演算によって実現される．$GF(2^n)$ 上では正負の区別をせず，その加算は $GF(P)$ 上と異なり，ビット同士の単純な XOR(排他的論理和) で定義される．従って，桁上げの考慮の必要がない $GF(2^n)$ では回路の遅延時間が桁上げの考慮の必要がある $GF(P)$ のものに比べて短い．ただし，楕円曲線暗号を用いた署名方式である EC-DSA[5] や IEEE の暗号規格 1363-2000[10] では $GF(P)$ と $GF(2^n)$ の両方で標準化規定があるため，両フィールドが扱えるユニファイド型アーキテクチャを持つ演算器が必要である．このユニファイド型では，遅延時間の異なる $GF(P)$ と $GF(2^n)$ の乗算器を統合する手法が必要になる．

本論文では，$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブル双基数ユニファイド型モンゴメリ乗算器を提案することを目的とする．提案アーキテクチャは基数 2^{16} で 4 並列化した $GF(P)$ 乗算器と基数 2^{64} の $GF(2^n)$ 乗算器を 1 つに統合するものである．双基数化によって $GF(2^n)$ と $GF(P)$ における遅延時間差を削減し，それに伴う低基数側のクロックサイクル数増加を，並列化によって削減する．4 個の $GF(P)$ 部分と $GF(2^n)$ 部分は大部分が共通化できるため，並列化による面積オーバヘッドが抑えられる．基数が小さい $GF(P)$ 計算時には必要クロックサイクル数が増えるが，同時にクリティカルパスを短く出来るため，$GF(2^n)$ と $GF(P)$ における遅延時間差が削減される．従って，提案手法では，$GF(P)$ 乗算器と $GF(2^n)$ 乗算器と同じ動作周波数で動作させることでき，モンゴメリ乗算全体としての演算実行時間が既存手法より短いスケーラブルユニファイド型モンゴメリ乗算器となる．

1.2　本論文の概要

本論文では，$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブル双基数ユニファイド型モンゴメリ乗算器を提案することを目的とする．以下に各章の概要を示す.

第 2 章「**公開鍵暗号**」では，公開鍵暗号方式の基本的な性質について示し，共通鍵暗号との違いを明確にする．楕円曲線暗号の安全性は離散対数問題に基づいた逆計算の困難さによるものであり，それが現在広く使用されている RSA 暗号よりも小さい鍵長で同等の暗号強度を実現していることを示す．また楕円曲線の安全な選び方について示す.

第 3 章「**楕円曲線暗号アルゴリズム**」では，楕円曲線暗号を実装するために必要なアルゴリズムについて示す．暗号に用いる楕円曲線の式について述べ，楕円曲線上での加法群について示す．加法群を実装するために必要な有限体上の演算，座標系，スカラー乗算について提案されている手法についても示し，比較する．

第 4 章「**モンゴメリ乗算アルゴリズム**」では，楕円曲線暗号の中で最も支配的な演算となる剰余乗算を，効率よく実行するモンゴメリ乗算のアルゴリズムを示す．最初に $GF(P)$ 及び $GF(2^n)$ におけるモンゴメリ乗算アルゴリズムを示し，次に，これらをスケーラブルに計算できるモンゴメリ乗算アルゴリズムを示す.

第 5 章「**既存モンゴメリ乗算器**」では，既存のモンゴメリ乗算器を示す．それぞれの既存モンゴメリ乗算器のアーキテクチャの分類を行い，各々の長所及び短所を示す.

第 6 章「**提案モンゴメリ乗算器**」では，$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブル双基数ユニファイド型モンゴメリ乗算器アーキテクチャを提案する．これを FPGA 上で実装検証を行い，その結果について示す．また，90nm プロセスライブラリを用いた論理合成による，面積と遅延時間の見積もり結果を示す．この論理合成結果に基づき，提案乗算器の遅延時間及び面積の評価を行い，既存手法との比較考察を行う.

第 7 章「**結論**」では本論文の内容についてまとめ，今後の課題を示す.

第2章

公開鍵暗号

2.1　本章の概要

本章では，公開鍵暗号の原理を示し，従来の共通鍵暗号との違いについて示す．また，公開鍵暗号の分野における楕円曲線暗号の位置付けを明確にする．楕円曲線暗号の安全性について示し，RSA 暗号との違いを示す．また楕円曲線の選び方を示す．

2.2 節「**暗号の基本**」では，暗号の基本的な事項について示し，共通鍵暗号と公開鍵暗号の比較について述べる．また，公開鍵暗号の原理を示す．

2.3 節「**楕円曲線暗号**」では，公開鍵暗号の歴史的背景について触れ，公開鍵暗号における楕円曲線暗号の位置付けを示す．

2.4 節「**暗号の安全性**」では，公開鍵暗号の安全性を示す．

2.5 節「**楕円曲線暗号の一般的安全性**」では，楕円曲線暗号の安全性について示し，RSA 暗号との違いを明らかにする．また，楕円曲線の選択方法について示す．

2.2　暗号の基本

暗号（cryptography）とは，第三者に知られたくない情報やデータを秘密にするための方法である．例えばある情報をやり取りする際に電話や手紙，電子メールを使ったとすると，その様なデータは第三者に盗聴される恐れのある通信路（communication channel）を使って送られる．それを他者に知られたくない時，データを暗号化をして第三者にやり取りの内容がわからないようにする必要がある．暗号を用いた通信システムの様子を図 2.1 に示す．

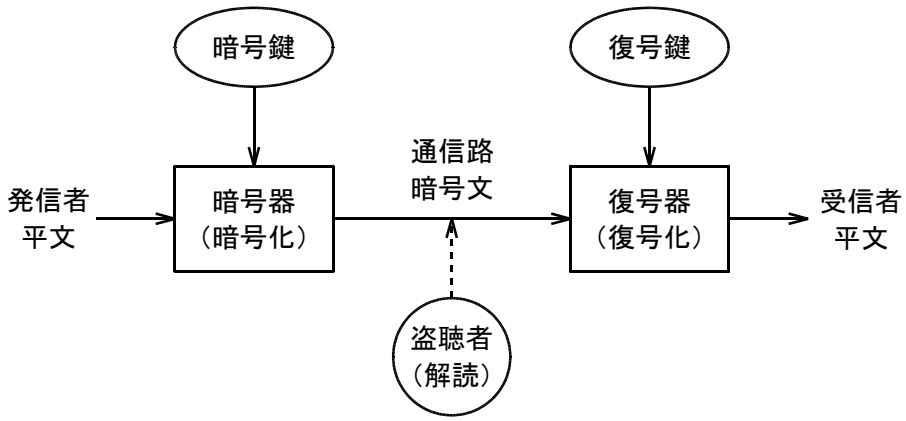

図 2.1: 暗号システム

発信者（sender）が受信者 (receiver）に伝えたい情報を平文（plaintext）といい，誰でもその意味や内容を理解できる．そのため発信者は平文を暗号化（encryption）して暗号文（ciphertext）にして理解できないようにする．受信者は暗号文を受け取り，復号（decryption）して平文を得る．盗聴者（wiretapper）が盗聴した暗号文を平文に戻すことを解読（cryptanalysis）という．暗号化したり復号したりするための情報を鍵（key）といい，特に，暗号化に用いる鍵を暗号鍵，復号に用いる鍵を復号鍵という．

暗号の原則としてアルゴリズムと鍵は区別して考える．暗号文は復号鍵が無いと復号できないが，それ以外の情報（アルゴリズムなど）は公開してもかまわない．秘密にする情報を鍵に限定することで暗号の取り扱いを容易にしている．

暗号の種類には大きく分けて共通鍵暗号（common-key cryptography）と公開鍵暗号（public-key cryptography）とがある．共通鍵暗号とは暗号化と復号化に

同じ鍵を用いる暗号である．歴史的に公開鍵暗号が登場するまでは暗号とは共通鍵暗号のことを指していた．共通鍵暗号は秘密鍵暗号（private-key cryptography）や対称鍵暗号（symmetric-key cryptography）とも呼ばれ，また公開鍵暗号は非対称鍵暗号とも呼ばれるが，本論文では共通鍵暗号，公開鍵暗号で統一する．共通鍵暗号には鍵配送（key distribution）という根本的な問題が存在する．暗号化・復号化の鍵が同じであるので，情報をやり取りするには送信者と受信者が鍵を共有する必要がある．またその鍵が第三者に渡ってしまえば容易に解読されてしまうため，鍵を他人に知られないようにする必要がある．そのため鍵のやりとりには盗聴される心配のない安全な通信路を使う必要がある．しかし現実には，安全な通信路を常に実現することは非常に難しい．距離が近いのなら直接会うなどの方法などがあるが，電話や電子メールといった電気通信路のような通信は，対称範囲が非常に広く，不特定多数とやり取りすることが多いため，安全な通信路の確保はきわめて困難である．

公開鍵暗号とは公開鍵と秘密鍵の異なる 2 つの鍵を使用する暗号方式である．公開鍵はその名の通り，世間に広く公開する鍵である．秘密鍵は秘密にしておき人に知られてはいけない鍵である．公開鍵と秘密鍵は対になっており，通信を行う際には送信者，受信者ともに鍵を持っている必要がある．暗号化・復号化をする場合には公開鍵と秘密鍵両方が必要になり，どちらか片方だけでは役に立たない．A を送信者，B を受信者とする．それぞれの秘密鍵と公開鍵の関係を表 2.1 に示す．秘密鍵 a_A, a_B は定数，G は公開されている共通の情報である．公開鍵は自身

表 2.1: 公開鍵暗号の秘密鍵と公開鍵の例

	秘密鍵	公開鍵
A（送信者）	a_A	$P_A = a_A \cdot G$
B（受信者）	a_B	$P_B = a_B \cdot G$

の秘密鍵と G を乗算したもので，簡単に計算できる．逆に公開鍵から秘密鍵を求めるのが極めて難しいので，公開しても解読される危険性は小さい．次に通信の手順を示す．

1. 送信者 A は受信者 B の公開鍵 P_B を入手

2. 平文 M を暗号化:$M' = a_A P_B + M$

3. 暗号文 M' を B に送信

4. B は A の公開鍵 P_A を入手

5. M' を復号化：$M' - a_B P_A = M$

ここで

$$a_A P_B = a_A a_B \cdot G \tag{2.1}$$

$$a_B P_A = a_B a_A \cdot G = a_A a_B \cdot G \tag{2.2}$$

よって

$$a_A P_B = a_B P_A \tag{2.3}$$

逆算すれば復号できることがわかる．公開鍵暗号を実現するために必要な要素は $P_A \leftarrow a_A \cdot G$ は簡単に計算できるが，$P_A/G = a_A$ を求めるのは非常に困難であるという性質である．この性質を実現する方法が大きな数の素因数分解の困難さや楕円曲線上の離散対数問題である．

公開鍵暗号は共通界暗号が持つ鍵配送の問題を解決し，さらに認証や署名機能といった新たな暗号の使い方を示した暗号方式である．提案されたのが 1976 年とその歴史は浅いため現代暗号とも言われている．情報化社会を支える基幹技術として注目されている．ちなみに鍵配送の問題が克服されたのでもう共通鍵方式は必要ないのかというとそうではない．公開鍵方式は鍵の生成に時間がかかり，実際には小さなデータしか扱うことができない．公開鍵暗号の用途は小さなデータで済むような使い方がメインである．認証や署名に用いたり，データ全体の暗号化は共通鍵方式で行い，その共通鍵を送るのに公開鍵方式を使う方法（Diffie-Hellman 鍵交換）などが提案されている．

2.3　楕円曲線暗号

公開鍵暗号の概念は，1976 年に Diffie と Hellman により提案された．Diffie-Hellman 鍵交換と呼ばれ，従来の共通鍵暗号における鍵配送問題を解決することを目的とした．Diffie と Hellman が提案した手法は概念的なものであり，実現方法は提案されていなかった．また第三者が送信者と受信者の間に割り込み、公開鍵を自分のものとすりかえて相手に気づかれることなく盗聴する中間者攻撃には弱いという問題点が指摘されていた．しかし，同時期に大きな数の素因数分解の困難性を利用した RSA 暗号 [24] が提案され，公開鍵が本物であることを証明する認証機能も実現できたため，中間者攻撃を防ぐことができ，Diffie-Hellman 鍵交換方式を実現することが可能となった．また公開鍵暗号方式には ElGamal 暗号 [6] というものも存在する．ElGamal 暗号は有限体の乗法群上で定義された離散対数問題の困難性を利用しているが，現実的な鍵長で安全性を確保することが難しく，実用化にはいたらなかった．しかし，1985 年に楕円曲線上で加法を定義し，その有限群上における離散対数問題で ElGamal 暗号を再構築する方法が Koblitz と Miller により独立に提案された [13, 19]．これが楕円曲線暗号である．またこの楕円曲線の離散対数問題は ElGamal 暗号だけでなく，Diffie-Hellman 鍵交換方式にも応用可能である．つまり楕円曲線暗号とは特定の 1 つの方式を意味するのではなく，楕円曲線 Diffie-Hellman 鍵交換方式や楕円曲線 ElGamal 暗号方式など楕円曲線の離散対数問題に基づき構成された暗号方式の総称であることに注意する [21]．楕円曲線暗号は現在の公開鍵暗号の中でデファクトスタンダードとなっている RSA 暗号と比べ，小さな鍵長で同等の安全性を実現できるとされている．そのため制約条件が厳しい組み込みシステムに適しているとされている．

2.4　暗号の安全性

暗号の歴史は，暗号方式を作る・解読するという繰り返しでであった．そのため暗号は単なる技法（art）の域を出なかったが，1980 年代初めより公開鍵暗号の安全性を理論的に定式化して，ある暗号方式が安全であることを証明しようという理論研究が盛んに行われてきた．一般に暗号の安全性は，その暗号を解読するのに必要な計算時間で評価される．ただし「暗号を解読するのに必要な計算時間」とは実測値ではない．実測できるということは，その暗号は解読されたことを意味しているため，そのような暗号は危険なので使用されない.「暗号を解読するために必要な計算時間」とは，ある攻撃方法を選択した場合の計算量から算出する期待値である．複数の攻撃法がある場合は，その中の最も計算時間の少ないものを用いてその暗号の安全性を評価する．計算時間が期待値であるための平均量であり，この評価方法では特殊な場合の安全性は評価できない．

2.5　楕円曲線暗号の一般的安全性

楕円曲線暗号の安全性は楕円曲線上の離散対数問題に拠っている．楕円曲線上の離散対数問題の解法は一般的離散対数問題の解法である平方根法と，曲線の位数を素因数分解して小さな離散対数問題に帰着させる PHS 法（Pohlig-Hellman-Silverman 法）を利用した攻撃法しか見つかっていない [21].

平方根法は，最も汎用的な解法で，その計算時間は鍵のビット長の半分のべき乗に比例する時間が必要である．公開鍵暗号の攻撃に必要な計算時間が指数時間であることは，非常に安全であるといわれている．

PHS 法は，曲線の位数が小さい素因数の積の形で表されるときに効率的に離散対数問題を解くことが可能になる．逆に位数が素因数分解されない場合にはその計算量は平方根と同等になる．楕円曲線暗号への攻撃に必要な計算時間は指数時間となり，非常に高い安全性が確保できることになる．

現在のデファクトスタンダードである RSA 暗号の安全性と比較する．RSA 暗号の安全性は大きな整数の素因数分解に拠っている．大きな素因数分解には数体ふるい法と呼ばれる効率的な攻撃法が存在し，そのため RSA 暗号への攻撃に必要な計算時間は準指数時間となる．準指数時間とは計算が容易とされる多項式時間と大変困難とされる指数時間の中間に位置されるもので，数対ふるい法の場合には鍵のビット長の 3 乗根のべき乗にほぼ比例する．したがって同じ鍵長の場合，指数時間より効率的に計算できる．1024 ビット RSA 暗号と 163 ビット楕円曲線暗号が同等の安全性をもつとされているのはこのためである（図 2.2）.

楕円曲線暗号は曲線によって効率的な攻撃方法が存在する．楕円曲線の位数の性質を表す指標の１つに trace がある．この trace の値が 0〜2 と小さい値の場合，その特性を利用した攻撃が存在し，それらを用いると非常に効率的に計算することが可能となる．trace=0 は超特異（supersingular）と呼ばれる曲線である．1985 年に Koblitz と Miller によって提案された楕円曲線暗号はこの超特異な楕円曲線を使うことを推奨していた．しかし 1990 年に超特異楕円曲線上の離散対数問題に対して準指数時間の解読法（MOV 帰着）が発見された．trace=1 の楕円曲線は anomalous 曲線と呼ばれているが，解読法（SSSA アルゴリズム）が 1997 年に発見された．この解法は，大変効率がよく多項式時間（サイズの 3 乗）である．trace=2

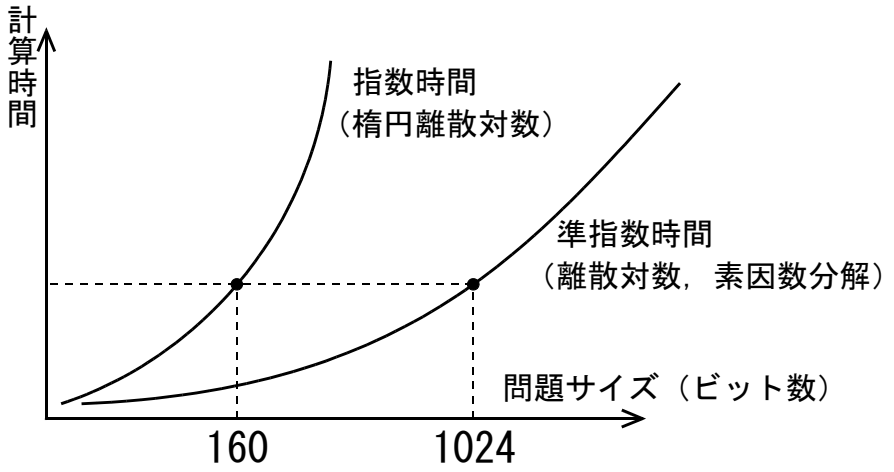

図 2.2: 計算時間と鍵サイズの比較

の楕円曲線は 1998 年に解読法（US アルゴリズム）があることが示されている．それ以外の trace については曲線をランダムに選ぶことで，現在わかっている手法により準指数時間で解読される曲線が選ばれる可能性はきわめて小さい．

楕円曲線暗号に使う安全な曲線として．

- 曲線の位数が素数，またはほぼ素数であること
- 特殊な曲線にならない（特に trace=0〜2 にはならない）

ことが必須である．

安全な曲線の生成では，現在 2 つの方法が提案されている．

- 曲線をランダムに選び，その位数を計算して，上記の条件を満たすものを探す方法
- あらかじめ条件を満たす良い位数を計算して，その位数になるような曲線を構成する方法

この分野における最も重要な未解決問題は，楕円曲線離散対数問題に対して指数計算法が適用できない本質的な理由があるのかどうかを明確にすることである．

2.6　本章のまとめ

本章では，公開鍵暗号の概要と楕円曲線暗号の安全性について示した．

2.2 節「**暗号の基本**」では，公開鍵暗号の具体的な暗号化方法を示し，従来の暗号である共通鍵暗号との違いを明確にした．共通鍵暗号は鍵の配送と鍵の生成という根本的な問題を持っており，公開鍵暗号はこの問題を解決した方法であることを示した．

2.3 節「**楕円曲線暗号**」では，公開鍵暗号が提案され，それが現在にいたるまでの過程を示し，楕円曲線暗号の公開鍵暗号に対する位置付けを明確にした．楕円曲線暗号は指数時間のオーダーを持つ離散対数問題であり，RSA 暗号と比べ少ない鍵長で同等の安全性を実現できる．そのため扱うデータが小さく済み，メモリやプロセッサの性能が制限される IC カードのようなシステムに適した方式といえる．

2.4 節「**暗号の安全性**」では，暗号の安全性について示した．解読できないことを証明するのではなく，攻撃方法から暗号を解読する計算量のオーダーを求め，どれだけ時間がかかるかで安全性を定義している．例えばオーダーが指数時間ならば最速のスーパーコンピュータを使っても，解読するのに何千年も時間がかかってしまう．

2.5 節「**楕円曲線暗号の一般的安全性**」では，楕円曲線暗号の安全性について考察した．楕円曲線暗号を解読するために必要な計算量は指数時間であり，RSA 暗号は準指数時間である．そのため楕円曲線暗号は RSA 暗号よりも小さい鍵長で同等の安全性を実現できることを示した．しかし，楕円曲線の選び方によっては準指数時間や多項式時間の解読法が見つかっており，安全な曲線を選ぶ必要があることを示した．

第3章

楕円曲線暗号アルゴリズム

3.1　本章の概要

楕円曲線暗号の安全性は楕円曲線上で定義された加法群の離散対数問題に拠っている．また，楕円曲線暗号を実装するには加法群を演算するアルゴリズムを求める必要がある．本章では，楕円曲線暗号の演算アルゴリズムについて示す．

3.2 節「**楕円曲線の式**」では，楕円曲線暗号に使う楕円曲線の式について示す．Weierstrass 曲線について示し，体 K によって式が異なることを示す．

3.3 節「**楕円曲線上の加法群**」では，楕円曲線上における加法群を定義し，加算と倍算のアルゴリズムと体 K による性質の違いについて示す．

3.4 節「**有限体上の演算**」では，実装に必要な四則演算についてコスト比較について示す．

3.5 節「**座標系**」では，加法群のアルゴリズムに用いるアフィン座標と射影座標について示す．

3.6 節「**スカラー乗算**」では，楕円曲線暗号における実行時間で支配的なスカラー乗算について示す．

3.7 節「**既存手法の比較**」では，既存のスカラー乗算アルゴリズムについて示し，演算量の比較を行う．

3.2　楕円曲線の式

楕円曲線暗号に用いる楕円曲線は，代表的なものとしてWeierstrass曲線とKoblitz曲線の2つがある．本章ではWeierstrass曲線について示す．体K[1]におけるWeierstrass曲線を式（3.1）に示す．

$$E : y^2 + a_1xy + a_3 = x^3 + a_2x^2 + a_4x + a_5 \tag{3.1}$$

$a_1, a_2, a_3, a_4, a_6 \in K$ かつ $\Delta \neq 0$ である．Δ はEの判別式で，$\Delta \neq 0$ は重根を持たないための条件である．Δ は次の式で表される．

$$\left.\begin{aligned} \Delta &= -{d_2}^2 d_8 - 8{d_4}^3 - 27{d_6}^2 + 9d_2d_4d_6 \\ d_2 &= {a_1}^2 + 4a_2 \\ d_4 &= 2a_4 + a_1a_3 \\ d_6 &= {a_3}^2 + 4a_6 \\ d_8 &= {a_1}^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2{a_3}^2 - {a_4}^2 \end{aligned}\right\} \tag{3.2}$$

体 $K = \mathbb{F}_P$ で P が素数のときこれを素体と呼び，$GF(P)$ と表す．また，$q = P^n$ のとなる体 $L = \mathbb{F}_q$ を体 K の拡大体といい，$GF(q) = GF(P^n)$ と表す．拡大体の場合でも標数[2]を体としたときと同じ性質を持つ．L を K の任意の拡大体とし，楕円曲線における体 L 上の有理点の集合は，

$$E(L) = \{(x, y) \in L \times L : y^2 + a_1xy + a_3 - x^3 - a_2x^2 - a_4x + a_6 = 0\} \cup \{\infty\} \tag{3.3}$$

ここで ∞ は無限遠点 $O = (\infty, \infty)$ を表す．

暗号で用いる場合，式（3.1）を体 K の標数によって簡略化して扱いやすくしている．楕円曲線暗号では標数が

- $char(K) = 2$
- $char(K) > 3$

の2つが用いられている．特に $char(K) = 2$ であるとき，つまり $GF(2^n)$ のときこれを2の拡大体もしくは2進体と呼ぶ．

[1] K の和集合に関する補集合と合併をとる演算に関して閉じている集合
[2] 体 $K = \mathbb{F}_{P^n}$ としたときの底 P のこと．char(K) とも表す．

char(K)=2 で，$a_1 \neq 0$ のとき，変数 x, y を以下のように変形できる．

$$(x, y) \to \left({a_1}^2 x + \frac{a_3}{a_1}, {a_1}^3 + \frac{{a_1}^2 a_4 + {a_3}^2}{{a_1}^3} \right)$$

式 E は

$$y^2 + xy = x^3 + ax^2 + b \tag{3.4}$$

となる．判別式は $\Delta = b$ となる．この曲線は特異点をもたない（non-supersinglar）曲線であり，楕円曲線暗号において秘密鍵が容易にわからない曲線である．

char(K)>3 の素数の場合，変数 x, y を以下のように変形できる．

$$(x, y) \to \left(\frac{x - 3{a_1}^2 - 12a_2}{36}, \frac{y - 3a_1 x}{216} - \frac{{a_1}^3 + 4a_1 a_2 - 12a_3}{24} \right)$$

式 E は

$$y^2 = x^3 + ax + b \tag{3.5}$$

となる．式 (3.4), 式 (3.5) はともに K の拡大体でも成り立つ．楕円曲線の定義をまとめると以下のようになる．

- 楕円曲線の式 E は体 K 上で定義される．係数 a, b も K の要素である．E が K 上で定義されているならば，K の拡大体でも E が定義できる．
- 式 E は判別式 $\Delta \neq 0$ の時に重根を持たない．
- 点 ∞ を無限遠点とする．

3.3　楕円曲線上の加法群

楕円曲線上の加法を定義する．楕円曲線上における有理点 A,B を加法した点 D は，図 3.1 に示す点 A,B を結ぶ直線が再び楕円曲線と交わる点 C の y 座標の符号を反転した点 D で定義される．このとき楕円曲線上の有利点は加法により群[3]をなす．わかりやすく言うと点 D は楕円曲線上の点としてかならず存在するということである．A=B の場合，曲線に接する接線となる（図 3.2）．D=A+A=2A と

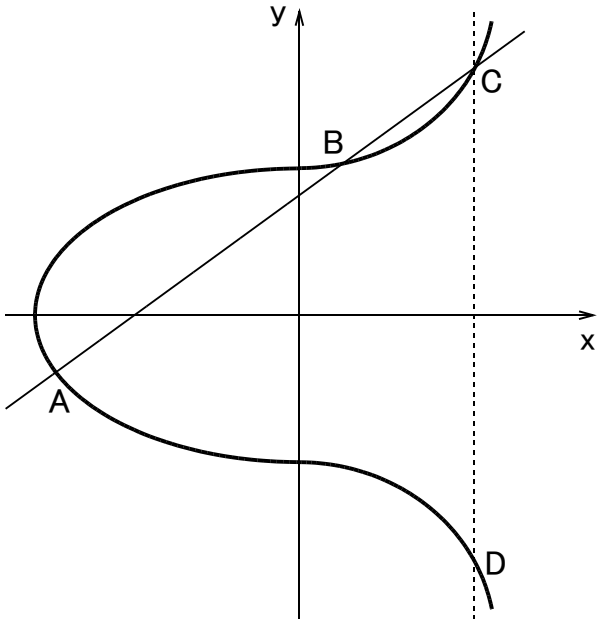

図 3.1: 楕円曲線上の加法（加算）

なる．加法の中で A≠B のときを点の加算，A=B のときを点の倍算という．

定義した加法を $E/\mathbb{F}_{2^n}$（式（3.4））， $E/K(char(K) > 3)$（式（3.5））に当てはめる．

- $E/\mathbb{F}_{2^n} : y^2 + xy = x^3 + ax^2 + b$ の加法群

  1. 加法の単位元：すべての $P \in E(\mathbb{F}_{2^n})$ において，$P + \infty = \infty + P = P$ が成り立つ．
  2. 負の数：$P \in E(\mathbb{F}_{2^n})$ において，$(x, y) = (x, x + y) = \infty$ が成り立つ．点 $(x, x + y)$ を $-P$ とし，P の負とする．また $-P \in E(\mathbb{F}_{2^n})$ である．

[3]結合則，単位元の存在，逆元の存在等公理系を満たす 2 項演算を持つ集合

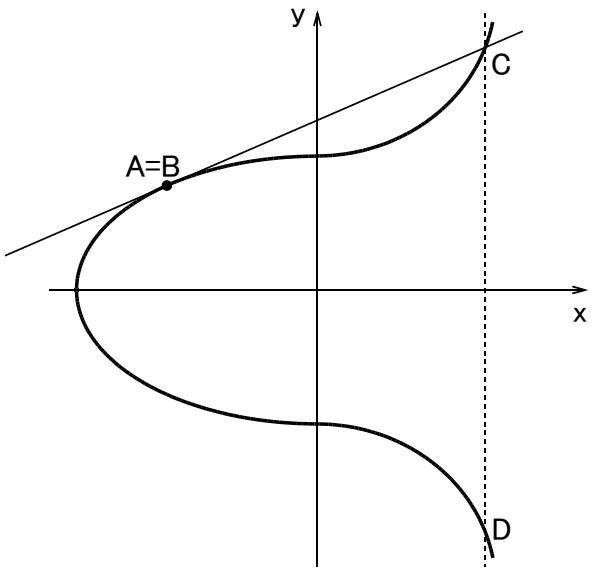

図 3.2: 楕円曲線上の加法（倍算）

3. 点の加算：$P = (x_1, y_1) \in E(\mathbb{F}_{2^n})$, $Q = (x_2, y_2) \in E(\mathbb{F}_{2^n})$ とし，$P \neq \pm Q$ とする．$P + Q = (x_3, y_3)$ とすると，

$$\left.\begin{aligned} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1 \end{aligned}\right\} \tag{3.6}$$

ただし，$\lambda = (y_1 + y_2)/(x_1 + x_2)$

4. 点の倍算：$P = (x_1, y_1) \in E(\mathbb{F}_{2^n})$ とし，$P \neq -P$ とする．$2P = (x_3, y_3)$ とすると，

$$\left.\begin{aligned} x_3 &= \lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2} \\ y_3 &= x_1^2 + \lambda x_3 + x_3 \end{aligned}\right\} \tag{3.7}$$

ただし，$\lambda = x_1 + y_1/x_1$

- $E/\mathbb{F}_p : y^2 = x^3 + ax + b, char(K) > 3$ の加法群

1. 加法の単位元：すべての $P \in E(\mathbb{F}_p)$ において，$P + \infty = \infty + P = P$ が成り立つ．
2. 負の数：$P \in E(\mathbb{F}_p)$ において，$(x, y) = (x, -y) = \infty$ が成り立つ．点 $(x, -y)$ を $-P$ とし，P の負とする．また $-P \in E(\mathbb{F}_p)$ である．

3. 点の加算：$P = (x_1, y_1) \in E(\mathbb{F}_p),\ Q = (x_2, y_2) \in E(\mathbb{F}_p)$ とし，$P \neq \pm Q$ とする．$P + Q = (x_3, y_3)$ とすると，

$$\left.\begin{aligned} x_3 &= \lambda^2 + x_1 + x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}\right\} \tag{3.8}$$

ただし，$\lambda = (y_2 - y_1)/(x_2 - x_1)$

4. 点の倍算：$P = (x_1, y_1) \in E(\mathbb{F}_p)$ とし，$P \neq -P$ とする．$2P = (x_3, y_3)$ とすると，

$$\left.\begin{aligned} x_3 &= \lambda^2 - 2x_1 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}\right\} \tag{3.9}$$

ただし，$\lambda = (3x_1^2 + a)/2y_1$

楕円曲線の重要な性質として有理点の数がある．$GF(q)$ 上で定義された楕円曲線 E を考える．E 上の有理点の集合を $E(\mathbb{F}_q)$ とし，有理点の数を $\#E(\mathbb{F}_q)$ と定義する．そのとき

$$q + 1 - 2\sqrt{q} \leq \#E(\mathbb{F}_q) \leq q + 1 + 2\sqrt{q} \tag{3.10}$$

が成り立つ．

3.4　有限体上の演算

楕円曲線上の加法群を実行するのは有限体上の演算である．実装する際に性能に大きな影響を与える重要な要素である．楕円曲線暗号の場合，特に効果的な体，すなわち素体，2 進体，拡大体の 3 つがある．その 3 つの体について簡潔に示す．まず有限体の一般的な性質を示す．

体 $\mathbb{F}$ には加算と乗算の 2 つがある．減算は $b+(-b)=0$ となる $\mathbb{F}$ の要素 $-b$ を b の負と定義し，$a,\ b \in \mathbb{F},\ a-b=a+(-b)$ として加算として行う．同様に除算は $b \cdot b^{-1}=1$ となる $\mathbb{F}$ の要素 b^{-1} を b の逆元と定義し，$a,\ b \in \mathbb{F},\ a/b=a \cdot b^{-1}$ として乗算として行う．

有限体上の演算を挙げると以下のようになる．

- 加算（Add）
- 乗算（Mul）
- 自乗（Sqr）
- 逆元計算（Inv）

自乗については，本来乗算と同じであるが体によって効率的なアルゴリズムが存在するのであえて別の演算とする．この 4 つの演算の中で重要なのが乗算と逆元演算である．加算は比較的単純なアルゴリズムで実装でき，乗算と比較するとコストが小さく無視できる場合が多い．自乗は 2 進体（$\mathbb{F}_{2^n}$）のとき効率的なアルゴリズムが存在し，乗算と比較した場合，そのコストは無視できるくらい小さくできる．その他の体の場合は無視できるほどではない．逆元計算は 4 つの中で最もコストが大きい．実際には複数回の乗算を行うことで計算される．有限体上の演算はさまざまなアーキテクチャが数多く提案されており，楕円曲線暗号に用いる際には実装の形態に合わせて選ぶ必要がある．

楕円曲線暗号のアルゴリズムを比較する方法として，有限体上の演算の回数を比較する手段がある．$Q=kP$ を求めるスカラー乗算のアルゴリズムは演算回数が一意に求まるので，簡単な比較が可能である．

3.5　座標系

楕円曲線暗号アルゴリズムに使われる座標系には，アフィン座標と射影座標の 2 つがある．アフィン座標はいわゆる x, y 座標のことを指す．第 3.3 節で示した加法群における点の加算と倍算の式（(3.6) , (3.7) , (3.8) , (3.9) はアフィン座標での計算式である．一方，楕円曲線上の点 (X, Y, Z) を定義すると，射影座標 $(X : Y : Z)$ は

$$(X : Y : X) = \{(\lambda^c X, \lambda^d Y, \lambda Z) : \lambda \in K\} \tag{3.11}$$

で定義される．$Z \neq 0$ のとき，対応するアフィン座標は $(X/Z^c, Y/Z^d, 1)$ となる．アフィン座標と射影座標は 1:1 に対応しており，相互に変換可能である．$Z = 0$ の場合，対応するアフィン座標は存在せず，射影座標における無限遠点 ∞ と定義する．また射影座標は $c,\ d$ の値によって特性が変わる．

- 標準的な射影座標 $(c = 1, d = 1)$
- Jacobian 座標 $(c = 2, d = 3)$
- LD 座標 $(c = 1, d = 2)$[4] [16]

射影座標の目的はアフィン座標での加法で必要な逆元計算をなくすことである．図 (3.3) に $\mathbb{F}_{2^n}$ における標準的な射影座標での点の加法のアルゴリズムを示す．点の加法が乗算のみできて，逆元計算を必要としない．ただし，点の加法をすべて終えアフィン座標で結果を与える必要があるときだけ逆元計算は 1 回必要になる．射影座標は逆元計算を除外できる一方で乗算の回数を増やしてしまうため，アフィン座標の加法に比べ高速に演算できるかは逆元演算が何回の乗算で実行するかの比 $I : M$ に依存する．$E/\mathbb{F}_p : y^2 = x^3 + ax + b$ における加法の演算回数の比較を表 3.1 [8] に，$E/\mathbb{F}_{2^n} : y^2 + xy = x^3 + ax^2 + b$ における加法の演算回数の比較を表 3.2 [8] にそれぞれ示す．混合加算とはアフィン座標 $(X_1, Y_1, 1)$ と射影座標 (X_2, Y_2, Z_2) の加算のことである．また表 3.2 の LD 座標は $a \in \{0, 1\}$ に限定することで加算のコストを減らしている [2].

[4] $\mathbb{F}_{2^n}$ の場合のみ

$$
\begin{array}{lll}
\lambda_1 & = X_1 Z_2^2 & 1M + 1S \\
\lambda_2 & = X_2 Z_1^2 & 1M + 1S \\
\lambda_3 & = \lambda_1 - \lambda_2 & \\
\lambda_4 & = Y_1 Z_2^3 & 2M \\
\lambda_5 & = Y_2 Z_1^3 & 2M \\
\lambda_6 & = \lambda_4 - \lambda_5 & \\
\lambda_7 & = \lambda_1 + \lambda_5 & \\
\lambda_8 & = \lambda_4 + \lambda_5 & \\
z_3 & = Z_1 Z_2 \lambda_3 & 2M \\
x_3 & = \lambda_6^2 - \lambda_7 \lambda_3^2 & 1M + 2S \\
\lambda_9 & = \lambda_7 \lambda_3^2 - 2X_3 & \\
y_3 & = (\lambda_9 \lambda_6 - \lambda_8 \lambda_3^3)/2 & 3M \\
\hline
 & & 12M + 4S
\end{array}
$$

図 3.3: 標数 $p > 3$ における射影座標での点の加法

表 3.1: 座標系による加法の演算コストの比較 $GF(P)$

座標系	加算	混合加算	倍算
アフィン座標	$1I + 2M + 1S$	-	$1I + 2M + 2S$
標準的な射影座標	$12M + 4S$	$8M + 3S$	$7M + 3S$
Jacobian 座標	$12M + 4S$	$8M + 3S$	$4M + 4S$

表 3.2: 座標系による加法の演算コストの比較 $GF(2^n)$

座標系	加算	混合加算	倍算
アフィン座標	$I + M$	-	$I + M$
標準的な射影座標	$13M$	$12M$	$7M$
Jacobian 座標	$14M$	$10M$	$5M$
LD 座標 $(a \in 0, 1)$	$14M$	$8M$	$4M$

3.6　スカラー乗算

本章では，定義した楕円曲線上の加法に基づいて $Q = kP$（k:整数，Q, P：楕円曲線上の点）を求める演算方法を示す．この演算をスカラー乗算といい，楕円曲線暗号の実行時間はこのスカラー乗算が支配的である [4]．スカラー乗算は基本的に楕円曲線の性質や座標系などの下位の要素に依存しないアルゴリズムである（一部例外あり）．スカラー乗算は用途によって複数の演算方法があるが，ここでは P を任意の入力とした場合の $Q = kP$ を求めるスカラー乗算に限定してそのアルゴリズムを示す．

3.6.1　二進展開法

スカラー乗算において最も単純な方法が二進展開法である．二進展開法には LSB から計算する方法と MSB から計算する場合の 2 種類が方法する．それぞれを図 3.4 [12] と図 3.5 に示す．

図 3.4: 二進展開法（Right to Left）

Input: $k = (k_{t-1}, \ldots, k_1, k_0)_2,\ P \in E(F_q)$.
Output: kP.

1. $Q \leftarrow \infty$.
2. For i from 0 to t-1 do
   2.1 If $k_i = 1$ then Q $\leftarrow$ Q+P.
   2.2 P $\leftarrow$ 2P.
3. Return(Q).

二進展開法は $t-1$ 回 の倍算と $W-1$ 回の加算（O を含むものは数えない）を必要とする．ここで t は二進展開法の長さであ W は重み（1 の数）である．

3.6.2　移動窓法

乗算 k のビットは長さ r のブロック（窓）の中で処理される．長さ r の窓を固定された桁数境界に合わせて処理を進めていき，それらに現れる 0 はスキップしている．このような実行の仕方は点の 2 倍計算の際に行われていたものである．移動窓法のアルゴリズムを図 3.6 に示す．

図 3.5: 二進展開法（Left to Right）

Input: $k = (k_{t-1}, \ldots, k_1, k_0)_2,\ P \in E(F_q)$.

Output: kP.

1. $Q \leftarrow \infty$.
2. For i from t-1 to 0 do
   2.1 Q $\leftarrow$ 2Q.
   2.2 If $k_i = 1$ then Q $\leftarrow$ Q+P.
3. Return(Q).

移動窓法を使うと，予備計算を増やすことなく 1 ビット大きい窓を設定したのと同等な効果を得る．この効果に関する直感的な説明は，k を表すビット列をコイン投げで表したときに，連続する移動窓の間にある 0 の示す「空白部分に」に長さ 1 を期待できることにある．したがって，処理される窓の総数（つまりメインループにおける一般の点の加算の総数）は $t(r+1)$ となる．

3.6.3　符号付表現（NAF）

式 $k = \sum_{j=0}^{t} s_j 2^j$，ここで $s_j \in \{-1, 0, 1\}$ の形の整数表現を（二進）符号付き桁 (SD:Signed Digit) 表示と呼ぶ．この表示方法は明らかに二進数表示を含み，また，すべての整数 $k, 0 \leq k \leq 2^{t+1} - 1$ もマイナス符号表現によって含まれている．しかし 3^{t+1} だけの可能な組み合わせがあるので，この表現が冗長なのは明らかである．たとえば整数 3 は $(001)_2$ とも $(10--1)_2$ とも表される．この冗長性はより実効的な空疎性の制約を持つスカラー乗算アルゴリズムに対してトレードを持つことがわかる．SD 表現が空疎であることは，隣接した 0 でないビットが存在しないこと，すなわち任意の $j \geq 0$ について $s_j s_{j+1} = 0$ なことである．空疎な SD 表現は非隣接形式 (NAF:non-adjacent form) と呼ばれる．NAF を使ったアルゴリズムを図 3.7 に示す.

3.6.4　Montgomery 型楕円曲線スカラー乗算

Montgomery スカラー乗算 [17] は $GF(2^n)$ 上の楕円曲線 $y^2 + xy = x^3 + ax^2 + b$ における LD 座標で演算するスカラー乗算である．Montgomery 法により x, z 座

図 3.6: 移動窓法

Input: $k = (k_{t-1}, \ldots, k_1, k_0)_2,\ P \in E(F_q)$.

Output: kP.

1. $P_1 \leftarrow P,\ P_2 \leftarrow 2P$.
2. For i from 1 to 2^{r-1} do
   2.1 $P_{2i+1} \leftarrow P_{2i-1} + P_2$.
3. $j \leftarrow t-1,\ Q \leftarrow \infty$.
4. For j from t-1 to 0 do
   4.1 If $k_j = 0$ then $Q \leftarrow 2Q$
   4.2 Else
      4.2.1 s を $j - s + 1 \leq$ と $k_s = 1$ である最小の整数とする.
      4.2.2 $h_j \leftarrow (k_j k_{j-1} \ldots k_s)_2$.
      4.2.3 $Q \leftarrow 2^{j-s+1}Q + P_{h_j}$.
5. Return(Q).

標のみ計算し，最後に y 座標を復元している．図 3.8 にアルゴリズムを示す.

N. Okeya [22] により，$GF(q)$ に一般化した Montgomery 型楕円曲線スカラー乗算が提案されている.

図 3.7: 二進 NAF 法

Input: 正の整数 k, $P \in E(F_q)$.

Output: kP.

1. $i \leftarrow 0$.
2. While $k \geq 1$ do
   2.1 If k is odd then
      2.1.1 $k_i \leftarrow (k \bmod 4)$.
      2.1.2 $k \leftarrow k - k_i$.
   2.2 Else $k_i \leftarrow 0$.
   2.3 $k \leftarrow k/2$.
   2.4 $i \leftarrow i + 1$.
3. NAF$(k) = (k_{i-1}, k_{i-2}, \ldots, k_1, k_0)$.
4. $Q \leftarrow \infty$.　5. For j from $l - 1$ downto 0 do
   5.1 $Q \leftarrow 2Q$.
   5.2 If $k_i = 1$ then $Q \leftarrow Q + P$.
   5.3 If $k_i = -1$ then $Q \leftarrow Q - P$.
6. Return(Q).

図 3.8: Montgomery 乗算

Input: $k = (k_{t-1}, \ldots, k_1, k_0)_2$ $P = (x, y) \in E(F_q)$.

Output: kP.

1. If $k = 0$ or $x = 0$ then output(0,0) and stop.
2. $X_1 \leftarrow x$, $Z_1 \leftarrow 1$, $X_2 \leftarrow x^4 + b$, $Z_2 \leftarrow x^2$.
3. For i from $l - 2$ downto 0 do
   - 3.1 If $k_i = 1$ then
     - 3.1.1 Madd(X_1, Z_1, X_2, Z_2).
     - 3.1.2 Mdouble(X_2, Z_2).
   - 3.2 Else
     - 3.2.1 Madd(X_2, Z_2, X_1, Z_1).
     - 3.2.2 Mdouble(X_1, Z_1).
4. Q=Mxy(X_1, Z_1, X_2, Z_2).
5. Return(Q).

3.7　既存手法の比較

各スカラー乗算の演算回数を比較する．楕円曲線上の加法群の加算 (A) と倍算 (D) の回数と有限体上の演算の乗算 (M) と逆元演算 (I) を示す．表 3.3 に $GF(192)$ での比較，表 3.4 に $GF(2^{163})$ での比較をそれぞれ示す．

どちらの場合も射影座標のほうが乗算回数が少なく有利に見えるが，逆元器 I により結果は大きく異なるため一概には言えない．Montgomery 型楕円曲線でのスカラー乗算は点を記憶するメモリが必要なく，それでいて乗算回数は移動窓法に匹敵するが，y 座標を計算していないため用途が限られる．y 座標を復元する方法もあるが演算量は増える．

表 3.3: スカラー乗算の比較 $GF(2^{192})$

				加法群		有限体上の演算		
スカラー乗算	座標系	w	メモリ数	A	D	M	I	$I/M = 80$
二進展開法	アフィン	-	0	95	191	977	286	23857
	射影	-	0	95	191	2420	1	2500
二進 NAF 法	アフィン	-	0	63	191	886	254	21206
	射影	-	63	0	191	2082	1	2162
移動窓法	アフィン	4	3	41	193	1840	4	2160
	射影	5	7	38	192	1936	1	2016

表 3.4: スカラー乗算の比較 $GF(2^{163})$

				加法群		有限体上の演算		
スカラー乗算	座標系	w	メモリ数	A	D	M	I	$I/M = 8$
二進展開法	アフィン	0	0	81	162	486	243	2430
	射影	0	0	81	162	1298	1	1306
二進 NAF 法	アフィン	0	0	54	162	432	216	2160
	射影	0	0	54	162	1082	1	1090
移動窓法	アフィン	4	3	35	163	396	198	1980
	射影	4	3	35	163	914	5	954
Montgomery 法	アフィン	-	0	162	162	328	325	2928
	射影	-	0	162	162	982	1	990

3.8　本章のまとめ

本章では，楕円曲線暗号のついて必要なアルゴリズムについて示した．

3.2 節「**楕円曲線の式**」では，楕円曲線暗号に使う楕円曲線の式について示した．Weierstrass 曲線はそのまま使うと複雑なので，体 K の標数によって式を簡略化できること示した．また拡大体でも楕円曲線の性質は変わらないことを示した．

3.3 節「**楕円曲線上の加法群**」では，楕円曲線上における加法群を定義し，加算と倍算のアルゴリズムと体 K による性質の違いについて示した．楕円曲線暗号で使われるのは $\mathrm{char}(K)$=2 の場合と，$\mathrm{char}(K)$>3 の素数の場合の 2 つである．それぞれの加法群の性質について示した．

3.4 節「**有限体上の演算**」では，実装に必要な四則演算についてコスト比較について示した．有限体上の演算は非常に多くの方法が提案されており，目的に合った実装方法を選ぶ必要がある．また四則演算の中で乗算と逆元演算のコストが大きく，この 2 つの扱いが重要である．

3.5 節「**座標系**」では，加法群のアルゴリズムに用いるアフィン座標と射影座標について示した．アフィン座標はいわゆる (x, y) 座標のことで加法群の加算と倍算を行うたびに逆元演算が必要である，それに対し射影座標はアフィン座標で必要が逆元演算がなくなるが，乗算回数が増える．逆元演算と乗算の比によって効率が変わる．

3.6 節「**スカラー乗算**」では，楕円曲線暗号における実行時間で支配的なスカラー乗算について示した．最も単純な二進展開法や，メモリを使って効率的に計算する移動窓法，符号付表現を使って計算する NAF 法，Montgomery 法などを示した．

3.7 節「**既存手法の比較**」では，スカラー乗算アルゴリズムについて 1 回の演算に必要な乗算の回数を示し，効率を比較した．二進展開法は単純でわかりやすいが演算量は多いほうである．移動窓法は効率は良いがその分メモリが多く必要である．NAF 法はメモリを使わなくても良い結果を出している．Montgomery スカラー乗算は演算回数は少ないが，y 座標計算を復元しない場合であり，復元する場合にはコストが増加する．

第4章

モンゴメリ乗算アルゴリズム

4.1　本章の概要

本章では，楕円曲線暗号の中で最も支配的な演算となる剰余乗算を，効率よく実行するモンゴメリ乗算のアルゴリズムを示す．最初に $GF(P)$ 及び $GF(2^n)$ におけるモンゴメリ乗算アルゴリズムを示し，次に，これらをスケーラブルに計算できるモンゴメリ乗算アルゴリズムを示す．

4.2 節「**$GF(P)$ におけるモンゴメリ乗算**」では，一般的な $GF(P)$ 上のモンゴメリ乗算のアルゴリズムを示す．

4.3 節「**$GF(2^n)$ におけるモンゴメリ乗算**」では，一般的な $GF(2^n)$ 上のモンゴメリ乗算のアルゴリズムを示す．

4.4 節「**スケーラブルモンゴメリ乗算**」では，4.2 節及び 4.3 節をスケーラブルに実行できるよう改変したアルゴリズムを示す．

4.2　$GF(P)$ におけるモンゴメリ乗算

一般的な $GF(P)$ 上のモンゴメリ乗算のアルゴリズムを図 4.1 に示す．ここで，n ビットの入力は m 個の d ビットのブロックに分割して表現するとする $(n = m \cdot d)$．大文字は n ビットの多倍長の値を表し，小文字は d ビットの 1 ブロックを表す．以後，本論文を通して使用する他の主な記号の定義は表 4.1 に示す．

INPUT: $A = (a_{m-1}, \cdots, a_1, a_0)_{2^d}, B, P \, (0 \leq A, B < P)$
INPUT: $q = -P^{-1} \bmod 2^d$
OUTPUT: $C = AB2^{-n} \bmod P$

$C := 0$
for $i = 0$ to $m - 1$
  $t_i := (c_0 + a_i b_0)\, q \bmod 2^d$
  $C := (c_i + a_i B + t_i P)/2^d$
if $(C > P)$ then $C := C - P$

図 4.1: $GF(P)$ におけるモンゴメリ乗算アルゴリズム．

表 4.1: 表記法

パラメータ	意味
A, B	モンゴメリ乗算のオペランド
a_i, b_i	A, B を構成する i 番目の d ビット (w ビット) のブロック
P	モンゴメリ乗算の法
q	$q = P^{-1} \bmod r$
S	モンゴメリ乗算中の部分積
C	モンゴメリ乗算結果
n	A, B, 及び P のビット長
r	基数 $(r = 2^d)$
d	d ビットブロックのビット長
m	d ビットブロックの数 $(m = \lceil n/d \rceil)$
w	w ビットブロックのビット長
e	w ビットブロックの数 $(e = \lceil n/w \rceil)$

モンゴメリ乗算のアルゴリズム (図 4.1) では，AB の乗算結果に法である P の倍数を加算し，下位 n ビットを 0 にすることで剰余演算を不要にしている．これに必要な q は事前に計算しておく．$m = 4$ のときを図式化したものを図 4.2 に示す．図 4.2 において C がモンゴメリ乗算の出力となる．

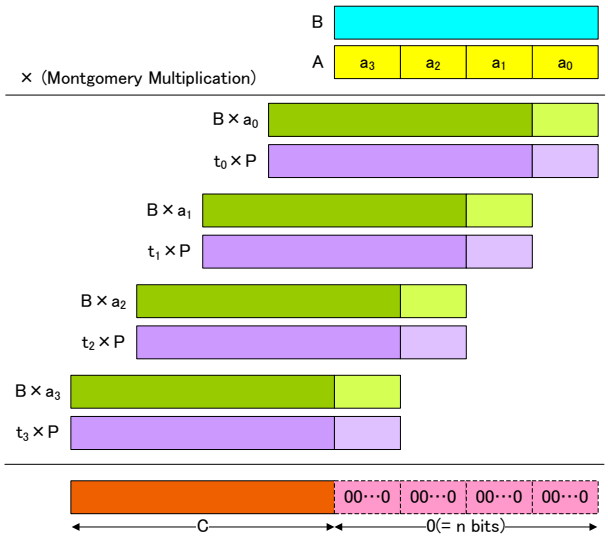

図 4.2: モンゴメリ乗算.

式 (4.1,4.2,4.3,4.4,4.5) を用いて，モンゴメリ乗算のアルゴリズム (図 4.1) の出力結果の下位 n ビットが 0 になることを示す．最初に図 4.1 中の一行

$$C := (c_i + a_i B + t_i P)/2^d \tag{4.1}$$

に注目する．この式のでは，下位 d ビットが 0 になることがわかっているので右に d ビットシフトしている．この下位 d ビットの 0 列は

$$(c_0 + a_i b_0 + t_i p_0) \bmod 2^d \tag{4.2}$$

によって与えられる．また，

$$t_i := (c_0 + a_i b_0)\, q \bmod 2^d \tag{4.3}$$

$$q = -P^{-1} \bmod 2^d \tag{4.4}$$

であることより，式 (4.2) は

$$\begin{aligned}(c_0 + a_i b_0 + t_i p_0) \bmod 2^d &= c_0 + a_i b_0 + (c_0 + a_i b_0)\left(-p^{-1}\right) p_0 \bmod 2^d \\ &= c_0 + a_i b_0 - (c_0 + a_i b_0) \bmod 2^d \\ &= 0 \end{aligned} \tag{4.5}$$

のように，0 列になることが示された．これを，m 回反復することで，モンゴメリ乗算結果の下位 n ビットが 0 になることが保障される ($n = m \cdot d$).

モンゴメリ乗算のアルゴリズム (図 4.1) の最終ステップで，減算が行われているが，これは桁上げを行った結果として $C = AB2^{-n}$ の値が $0 \leq C < 2P$ にあるからである．

4.3　$GF(2^n)$ におけるモンゴメリ乗算

$GF(2^n)$ 上でも $GF(P)$ の図 4.1 とほぼ同じ操作で演算が定義される．図 4.3 に $GF(2^n)$ 上における一般的なモンゴメリ乗算のアルゴリズムを示す．ただし，$GF(2^n)$ では $GF(P)$ と異なり，桁上げ作業が無いために，最終ステップで減算が必要になる可能性がない．

INPUT: $A(x) = (a_{m-1}, \cdots, a_1, a_0)_{x^d}, B(x), P(x)$
INPUT: $q(x) = P(x)^{-1} \bmod x^d$
OUTPUT: $C(x) = A(x) B(x) x^{-n} \bmod P(x)$

$C(x) := 0$
for $i = 0$ to $m - 1$
  $t_i(x) := [c_0(x) + a_i(x) b_0(x)] q(x) \bmod x^d$
  $C(x) := [c_i(x) + a_i(x) B + t_i(x) P(x)] / x^d$

図 4.3: $GF(2^n)$ におけるモンゴメリ乗算アルゴリズム．

4.4　スケーラブルモンゴメリ乗算

スケーラブルなモンゴメリ乗算を行うためには図 4.1 及び図 4.3 のアルゴリズムをそのまま用いることは出来ない．オペランド A は d ビットごとに区切られているので，n が変化したときは d は固定したまま m を増減させればよい $(n = d \cdot m)$．また，d は基数のビット幅を意味する．しかし，オペランド B は n ビットで，n ビット $\times d$ ビット乗算器を用いるため，n が変化したときに対応できない．対して，文献 [9, 11, 25, 27, 28, 32–34] ではオペランド B を w ビットごとの e 個に区切り，w ビット $\times d$ ビット乗算器を繰り返し使うアルゴリズムとアーキテクチャでスケーラビリティを持たせている．n が変化したときは m と同様に e も増減させる $(n = w \cdot e)$．

一般的に基数を大きくすると，乗算器の遅延時間は長くなる．文献 [9, 27, 28, 32–34] のアーキテクチャでは遅延時間を小さくする目的から基数のビット幅である d を 1 から 3 と低基数にしている．基数が小さいとクロックサイクル数は多くなるが，w を d 以上の大きさにすることと，1 演算器を 1 ステージとし，ステージを直列に並べて演算器の数を増やすことでクロックサイクル数を削減している．ただし，d が小さいうちはモンゴメリ乗算全体の演算時間を短縮することは難しい．反対に，d 及び w の両方を大きく (m，e を小さく) すればモンゴメリ乗算の必要クロックサイクル数は減る．従って，文献 [9, 11, 25, 27, 28, 32, 34] の中で最も高速なモンゴメリ乗算器は，$d = w = 64$ と最も高基数かつユニファイド型の文献 [25] の手法である．

文献 [25] のアーキテクチャでは 64 ビット $\times 4$ 入力・128 ビット出力の積和和演算器 (2 入力乗算結果に別の 2 つのオペランドを加算) を用いる．この積和和演算器を用いて，モンゴメリ乗算を行うアルゴリズムとして文献 [25] では，文献 [1, 14] で提案されている”finely integrated operand scanning method” を改変したアルゴリズムを用いている．文献 [25] で用いられた $GF(P)$ におけるスケーラブルモンゴメリ乗算アルゴリズムを図 4.4 に，$GF(2^n)$ におけるスケーラブルモンゴメリ乗算アルゴリズムを図 4.5 に示す．

INPUT: $A = (a_{m-1}, \cdots, a_1, a_0)_{2^d}$
INPUT: $B = (b_{m-1}, \cdots, b_1, b_0)_{2^d}$
INPUT: $P = (p_{m-1}, \cdots, p_1, p_0)_{2^d}\ (0 \leq A, B < P)$
INPUT: $q = -P^{-1} \bmod 2^d$
OUTPUT: $C = AB2^{-n} \bmod P$

$C := 0$
for $i = 0$ to $m - 1$
  $z := 0$
  $t_i := (c_0 + a_i b_0)\, q \bmod 2^d$
  for $j = 0$ to $m - 1$
    $S := c_j + a_i b_j + t_i p_j + z$
    if $(j \neq 0)$ then $c_{j-1} := S \bmod 2^d$
    $z := S/2^d$
  $c_{m-1} := z$
if $(C > P)$ then $C := C - P$

図 4.4: $GF(P)$ におけるスケーラブルモンゴメリ乗算アルゴリズム.

INPUT: $A(x) = (a_{m-1}, \cdots, a_1, a_0)_{x^d}$
INPUT: $B(x) = (b_{m-1}, \cdots, b_1, b_0)_{x^d}$
INPUT: $P(x) = (p_{m-1}, \cdots, p_1, p_0)_{x^d}$
INPUT: $q(x) = P(x)^{-1} \bmod x^d$
OUTPUT: $C(x) = A(x)\, B(x)\, x^{-n} \bmod P(x)$

$C(x) := 0$
for $i = 0$ to $m - 1$
  $z(x) := 0$
  $t_i(x) := [c_0(x) + a_i(x)\, b_0(x)]\, q(x) \bmod x^d$
    for $j = 0$ to $m - 1$
    $S(x) := c_j(x) + a_i(x)\, b_j(x) + t_i(x)\, p_j(x) + z(x)$
    if $(j \neq 0)$ then $c_{j-1}(x) := S(x) \bmod x^d$
    $z(x) := S(x)/x^d$
  $c_{m-1}(x) := z(x)$

図 4.5: $GF(2^n)$ におけるスケーラブルモンゴメリ乗算アルゴリズム.

4.5　本章のまとめ

本章では，楕円曲線暗号の中で最も支配的な演算となる剰余乗算を，効率よく実行するモンゴメリ乗算のアルゴリズムを示した．最初に $GF(P)$ 及び $GF(2^n)$ におけるモンゴメリ乗算アルゴリズムを示し，次に，これらをスケーラブルに計算できるモンゴメリ乗算アルゴリズムを示した．

4.2 節「**$GF(P)$ におけるモンゴメリ乗算**」では，一般的な $GF(P)$ 上のモンゴメリ乗算のアルゴリズムを示した．

4.3 節「**$GF(2^n)$ におけるモンゴメリ乗算**」では，一般的な $GF(2^n)$ 上のモンゴメリ乗算のアルゴリズムを示した．

4.4 節「**スケーラブルモンゴメリ乗算**」では，4.2 節及び 4.3 節をスケーラブルに実行できるよう改変したアルゴリズムを示した．

第5章

既存モンゴメリ乗算器

5.1　本章の概要

本章では，既存のモンゴメリ乗算器を示す．それぞれの既存モンゴメリ乗算器の各々の長所及び短所を示す．本章では，既存のモンゴメリ乗算器を示す．それぞれの既存モンゴメリ乗算器のアーキテクチャの分類を行い，スケーラブルかつ高基数で $GF(2^n)$ 及び $GF(P)$ の両フィールドが扱えるユニファイド型のモンゴメリ乗算器が必要であることを示す．また，$GF(2^n)$ と $GF(P)$ の両回路は遅延時間差が大きいため，$GF(P)$ での計算において動作周波数を落とすか，基数が $GF(2^n)$ 部分に比べて小さい双基数アーキテクチャにしなければならないことを示す．加えて，単純に $GF(P)$ 計算での基数を小さくすると必要クロックサイクル数が増えてしまうことを示す．

5.2 節「**スケーラブルモンゴメリ乗算器**」では，楕円曲線暗号ハードウェア向けスケーラブルモンゴメリ乗算器の利点を示し，既存の研究におけるスケーラブルモンゴメリ乗算器を示す．

5.3 節「**ユニファイド型モンゴメリ乗算器**」では，楕円曲線暗号ハードウェア向けユニファイド型モンゴメリ乗算器の利点と問題点を示し，既存の研究におけるユニファイド型モンゴメリ乗算器を示す．

5.2　スケーラブル乗算器

過去に提案されている楕円曲線暗号向けモンゴメリ乗算器および，モンゴメリ乗算器を用いた楕円曲線暗号 LSI には [4, 7, 9, 11, 18, 23, 25, 27, 28, 30, 32–35] が上げられる．しかし，これら全てのモンゴメリ乗算器がスケーラビリティという観点から楕円曲線暗号に適しているとはいえない．

楕円曲線暗号の暗号強度は鍵長のビット幅によって決定され，米国の国立標準技術研究所 (National Institute of Standards and Technology, NIST)[5] では 160 ビットから 256 ビットもしくはそれ以上を推奨している．従って，暗号強度によって，モンゴメリ乗算器のオペランドのビット幅も 160 ビットから 256 ビットもしくはそれ以上となる．それぞれのオペランドに対して，対応するビット幅の異なる乗算器を用意していてはハードウェアコストがかかる．そこで，スケーラブルな乗算器が望まれる．オペランドをある単位ビット幅ごとに分割し，その分割されたオペランドを共通の乗算器で反復的に演算すれば面積を削減できるためである．ここで，オペランドを分割する値を基数といい，この基数は表 4.1 における $r = 2^d$ に当たる．また，この d が図 4.2 における a_0,a_1,a_2,a_3 のビット幅に当たる．一般的に，基数のビット幅の大きく (高基数化) すれば 1 回のモンゴメリ乗算に必要なクロックサイクル数は削減できるが，乗算器の遅延時間は増す．図 5.1 に [25] における d と 1 回のモンゴメリ乗算に必要なクロックサイクル数の関係を示す．d と乗算器の遅延時間は第 6 章で述べる．

スケーラブルなモンゴメリ乗算のアルゴリズム及びアーキテクチャは [4, 7, 9, 11, 23, 25, 27, 28, 32–34] によって提案されている．これらの文献については 5.3 節において示す．一方 [18, 35] はスケーラブルな設計となっておらず，[5] の基準を満たせない．また，[23] はアーキテクチャの提案のみに留まり，実装に至っていないため，その有効性を確認することが出来ない．従って，[18, 23, 35] 今後の議論から除外する．

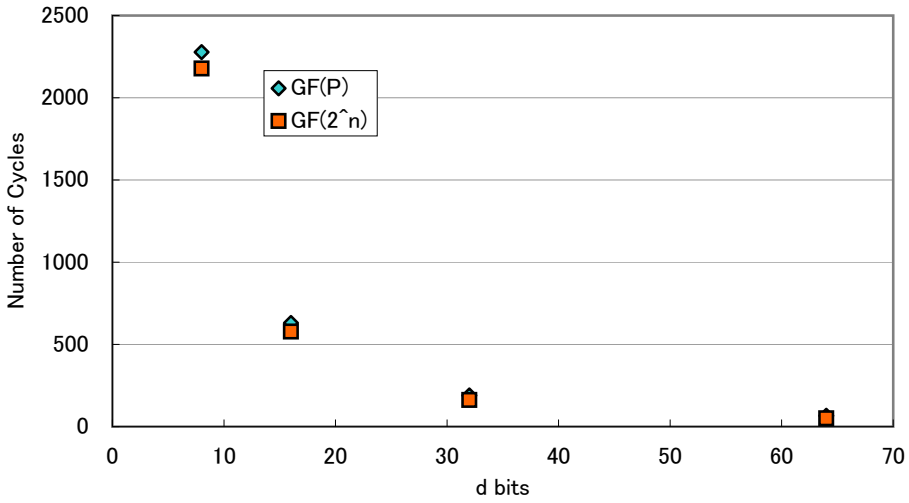

図 5.1: d と 1 回のモンゴメリ乗算に必要なクロックサイクル数の関係.

5.3　ユニファイド型乗算器

楕円曲線暗号は$GF(P)$もしくは$GF(2^n)$上で演算されるので，モンゴメリ乗算器も$GF(P)$と$GF(2^n)$上の両方の値を扱える必要がある．$GF(2^n)$上では正負の区別をせず，その加算は$GF(P)$上と異なり，ビット同士の単純なXOR(排他的論理和)で定義される．従って，桁上げの考慮の必要がない$GF(2^n)$では回路の遅延時間が$GF(P)$のものに比べて短い．ただし，楕円曲線暗号を用いた署名方式であるEC-DSA[5]やIEEEの暗号規格1363-2000[10]では$GF(P)$と$GF(2^n)$の両方で標準化規定があるため，両フィールドが扱えるユニファイド型アーキテクチャが[9, 25, 27, 28, 30]にて提案されている．[11, 32–34]は$GF(P)$に特化しているため，$GF(2^n)$が扱えず，[5, 10]の標準化規定に則ることが出来ない．同様に，[4, 7]は$GF(2^n)$に特化しているため，$GF(P)$が扱えず，[5, 10]の標準化規定に則ることが出来ない．

図5.2にユニファイド型乗算器のブロック図を示す．$GF(P)$及び$GF(2^n)$の両フィールドでまず$GF(2^n)$乗算器を用いて演算する．その後に，$GF(P)$ Carry Propagation Partを通過させることで桁上げを計算し，$GF(P)$としての乗算結果が得られる．この$GF(P)$ Carry Propagation Partの通過することで，遅延時間差が発生する．従って，$GF(P)$における乗算器の遅延時間は$GF(2^n)$における乗算器の遅延時間よりも常に長くなる．

文献[9, 27]は片方のオペランドを一度に1bitずつ処理する基数(radix)2を用いたアーキテクチャを提案している．この乗算器では$GF(P)$と$GF(2^n)$の共通部分を統合し，一つの演算器で両フィールドのモンゴメリ乗算が実行できる．しかし，基数2ではオペランドのビット数の2乗に比例してクロックサイクル数は増加するため，スループットを上げるのが難しい．

文献[25]では，64-bit×64-bit乗算器を用いたアーキテクチャが提案されている．基数を2^{64}と高基数化したことで，スループットを向上している．ただし，この乗算器では両フィールドの$GF(P)$部と$GF(2^n)$部の遅延時間差が大きく，$GF(2^n)$では$510.2MHz$で動作できるところを，$GF(P)$では$137.7MHz$までプロセッサ全体の周波数を落とす必要がある．従って，フィールドを切り替えるために，複数の動作周波数に対応した設計にする必要がある．

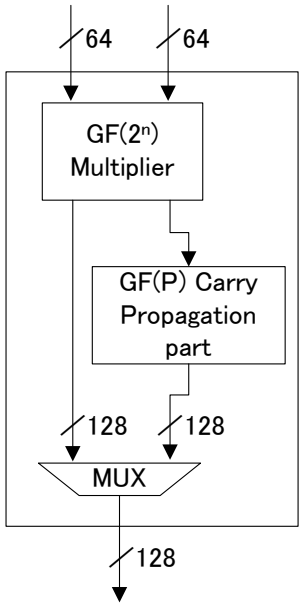

図 5.2: ユニファイド型乗算器のブロック図.

遅延時間差の問題に対し，文献 [28] では双基数アーキテクチャを提案し，遅延時間差に対応している．基数を小さくすると遅延時間が短くなることを利用して，文献 [28] の手法は，$GF(P)$ の基数を 2，$GF(2^n)$ の基数を 4 と双基数にすることで，遅延時間差を削減した．しかし，文献 [28] の手法では，$GF(P)$ 乗算の基数を 2 と小さくしたため必要クロックサイクル数が増える．よって，モンゴメリ乗算全体としての演算実行時間は文献 [25] に及ばない．

文献 [30] では FPGA に特化させるためビットスライス形の乗算器アーキテクチャを提案している．しかし，遅延時間を $GF(P)$ 側で合わせているため，$GF(2^n)$ 側で性能を引き出せていない．

5.4　本章のまとめ

本章では，既存のモンゴメリ乗算器を示した．それぞれの既存モンゴメリ乗算器の問題点から，スケーラブルかつ高基数で $GF(2^n)$ 及び $GF(P)$ の両フィールドが扱えるユニファイド型のモンゴメリ乗算器が必要であることを示した．また，$GF(2^n)$ と $GF(P)$ の両回路は遅延時間差が大きいため，$GF(P)$ での計算において動作周波数を落とすか，基数が $GF(2^n)$ 部分に比べて小さい双基数アーキテクチャにしなければならないことを示した．加えて，単純に $GF(P)$ 計算での基数を小さくすると必要クロックサイクル数が増えてしまうことを示した．

5.2 節「**スケーラブルモンゴメリ乗算器**」では，楕円曲線暗号ハードウェア向けスケーラブルモンゴメリ乗算器の利点を示し，既存の研究におけるスケーラブルモンゴメリ乗算器を示した．

5.3 節「**ユニファイド型モンゴメリ乗算器**」では，楕円曲線暗号ハードウェア向けユニファイド型モンゴメリ乗算器の利点と問題点を示し，既存の研究におけるユニファイド型モンゴメリ乗算器を示した．

第6章

提案モンゴメリ乗算器

6.1　本章の概要

本章では，$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブル双基数ユニファイド型モンゴメリ乗算器アーキテクチャを提案する．これを FPGA 上で実装検証を行い，その結果について示す．また，90nm プロセスライブラリを用いた論理合成による，面積と遅延時間の見積もり結果を示す．この論理合成結果に基づき，提案乗算器の遅延時間及び面積の評価を行い，既存手法との比較考察を行う．

6.2 節「**提案モンゴメリ乗算器アーキテクチャ**」では，$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブル双基数ユニファイド型モンゴメリ乗算器アーキテクチャを提案する．

6.3 節「HDL **による実装**」では，6.2 節にて提案したモンゴメリ乗算器を HDL により実装し，FPGA 上での実装検証及び 90nm プロセスライブラリを用いた論理合成結果結果について示す．

6.4 節「**性能評価**」では，6.3 節の論理合成結果に基づき，提案乗算器の遅延時間及び面積の評価を行い，既存手法との比較考察を行う．

6.2　提案モンゴメリ乗算器アーキテクチャ

入力ビット幅の等しい $GF(P)$ 乗算器と $GF(2^n)$ 乗算器では，桁上げ作業が必要である $GF(P)$ 乗算器の遅延時間の方が長くなる [31]．$GF(P)$ 乗算器と $GF(2^n)$ 乗算器の遅延時間差を図 6.1 に示す．$d = 64$ のとき，$GF(P)$ 乗算器と $GF(2^n)$ 乗算器の遅延時間差は 3.7 倍になる．この遅延時間は，VHDL を用いて記述した $GF(P)$ と $GF(2^n)$ の両乗算器を，Synopsys 社の Design Compiler で論理合成した結果によるものである．ライブラリには STARC90nm プロセス用ライブラリ[1]を用いている．図 6.1 では，d が大きくなるほど，遅延時間差も広がるが，この時の $GF(P)$ と $GF(2^n)$ の遅延時間差を削減できれば，両フィールドを扱うときに分周器などで複数の動作周波数対応させることを考慮する必要がなくなる．$d = 64$ のときの $GF(2^n)$ 乗算器による遅延時間とほぼ同じ $GF(P)$ 乗算器の遅延時間の d の値は 16 のときである．

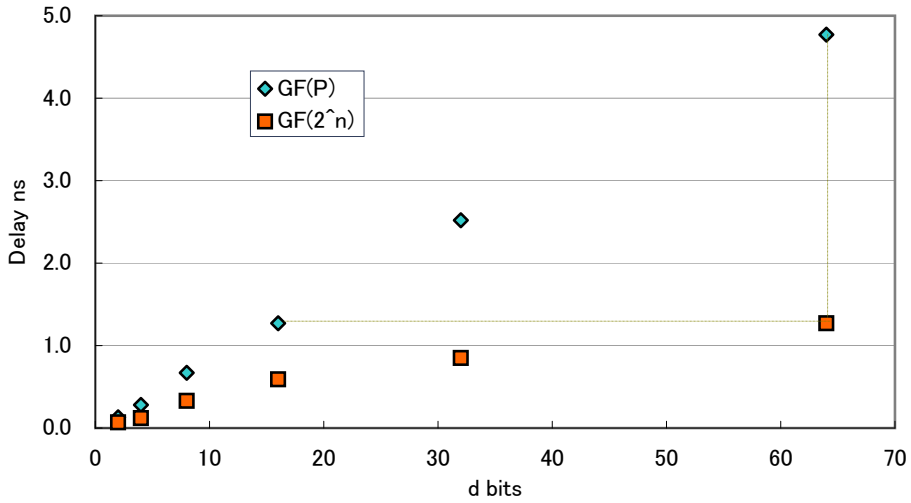

図 6.1: $GF(P)$ 乗算器と $GF(2^n)$ 乗算器の遅延時間差.

$GF(P)$ 乗算器と $GF(2^n)$ 乗算器の遅延時間差削減のために，基数 2^{16} で 4 並列化した $GF(P)$ 部分と基数 2^{64} の $GF(2^n)$ 部分を 1 つに統合した乗算アーキテクチャを提案する．

[1]STARC90nm プロセス用ライブラリは東京大学大規模集積システム設計教育研究センターを通し株式会社半導体理工学研究センターの協力で作成されたものである．

$GF(P)$ 計算時には $GF(2^n)$ との遅延時間差を削減する為に $GF(2^n)$ の基数よりも小さい基数を $GF(P)$ に用いて双基数化し，そのために増加するサイクル数を並列化により削減する．図 6.1 より，$d = 64$ のときの $GF(2^n)$ 乗算器による遅延時間とほぼ同じ $GF(P)$ 乗算器の遅延時間の d の値は 16 のときであった．従って，提案アーキテクチャでは，$GF(2^n)$ の乗算基数として 2^{64} を，$GF(P)$ の乗算基数として 2^{16} を選択する．演算器自体には 64bits の入力があるため，$d = 16$ の $GF(P)$ 乗算器を 4 並列に出来る．図 6.2 に 4 並列化した $GF(P)$ 部分と $GF(2^n)$ 部分を統合した乗算器のブロック図を示す．$GF(P)$ 計算時は 4 つの独立した 16-bit 乗算器として演算し，$GF(2^n)$ 計算時には 1 つの 64-bit 乗算器となる．$GF(P)$ 部分と $GF(2^n)$ 部分の遅延時間がほぼ等しいため同じ動作周波数で，演算を実行できる．

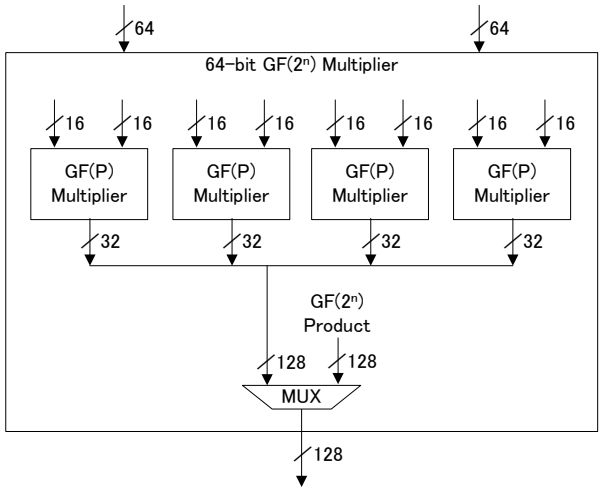

図 6.2: 双基数ユニファイド型乗算器.

図 6.3 に提案モンゴメリ乗算器を含んだ $GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号専用ハードウェアブロック図を示す．図 6.3 中の ALU では，図 4.4 及び，図 4.5 のアルゴリズムによるモンゴメリ乗算に必要な積和和演算，加減算，論理積演算が実行できる．図 6.3 中の ALU 内の双基数ユニファイド型乗算器部分を図 6.4 に示す．図 6.4 では 64bits の入力 A，B より 128bits の $GF(2^n)$ 乗算結果と 32bits の $GF(P)$ 乗算結果 4 つが出力される．外部からの制御信号によってどちらかのフィールドの結果を選択させる．各乗算結果に別の 2 つのオペランドを加算すれば，提案モンゴメリ乗算アルゴリズムに必要な積和和演算器が構成される．図 6.4

は両フィールドの共通部分を統合したユニファイド型乗算器 4 個と $GF(2^n)$ 部分だけの $GF(2^n)$ 乗算器 12 個で構成されている．図 6.4 中の両フィールドの共通部分を統合した 16-bit ユニファイド型乗算器を図 6.5 に示す．また，図 6.5 中の 4-bit ユニファイド型乗算器を図 6.6 に示す．

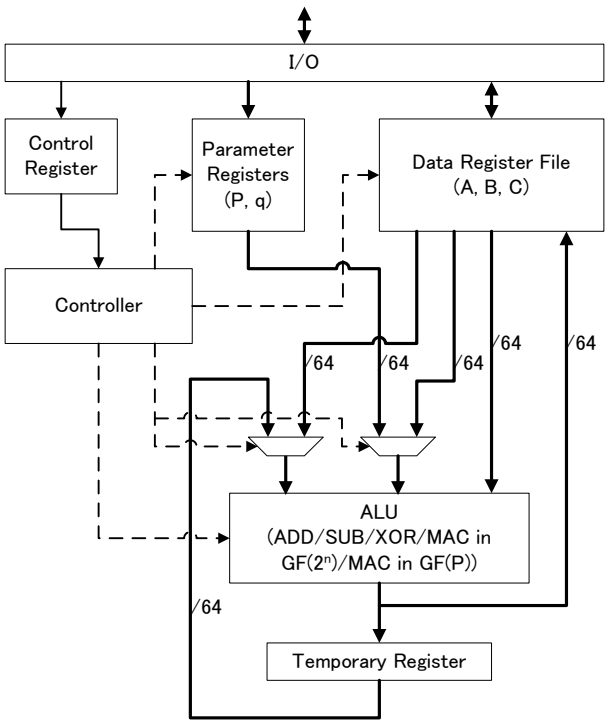

図 6.3: 提案モンゴメリ乗算器を含んだ $GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号専用ハードウェアブロック図.

$GF(2^n)$ における提案手法のクロックサイクル数見積もりは式 (6.1) で表される．ただし，1 回の積和和演算は 1 サイクルで実行される．

$$2m^2 + 5m + 3 \tag{6.1}$$

同様に，$GF(P)$ における提案手法のクロックサイクル数は式 (6.2) で表される．式 (6.2) は図 4.4 の最終ステップで減算が行われた最悪の場合である．

$$\frac{m^2}{2} + 3m + 6 \tag{6.2}$$

提案アーキテクチャより基数を 2^{16} とし，式 (6.2) から図 4.4 のアルゴリズムで並列化なしのモンゴメリ乗算のときと 4 並列化したモンゴメリ乗算のクロックサ

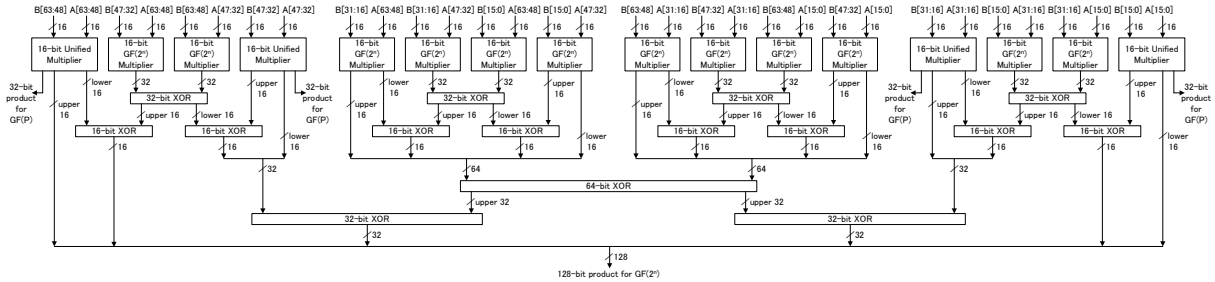

図 6.4: 64 ビット双基数ユニファイド型乗算器.

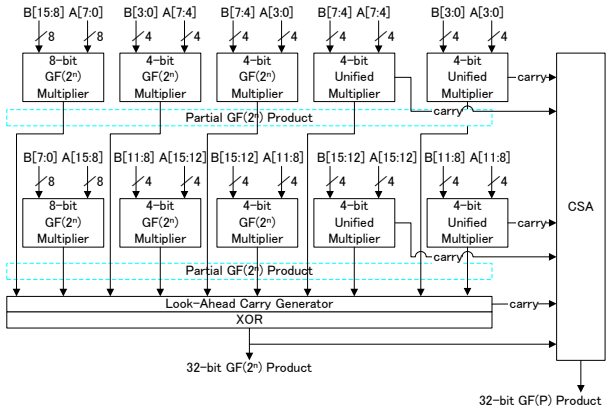

図 6.5: 16 ビットユニファイド型乗算器.

イクル数を表 6.1 に示す．並列化をせずに基数 2^{16} で 256 ビットモンゴメリ乗算を行うと，603 サイクルが必要になるが，4 並列化したことで，182 サイクル数まで削減される．並列化後に必要クロックサイクル数は並列化前の約 30% である．

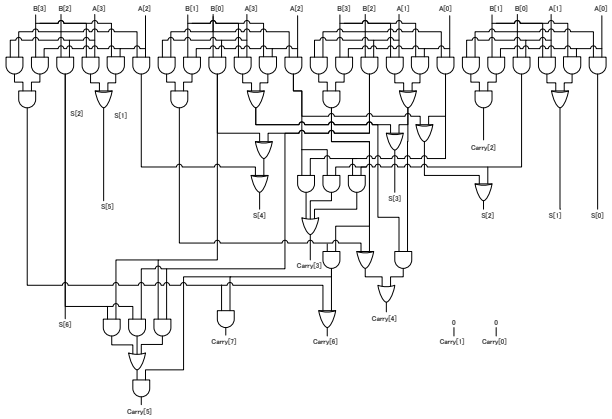

図 6.6: 4 ビットユニファイド型乗算器.

表 6.1: $GF(P)$ における基数 2^{16} モンゴメリ乗算のクロックサイクル数.

n (bits)	クロックサイクル数	
	並列化なし	4 並列化
160	258	86
192	357	114
224	472	146
256	603	182

6.3　HDL による実装

6.3.1　FPGA を用いた実装

6.2 節で提案したアーキテクチャは STARC90nm プロセス用ライブラリを用いることを前提としている．ただし，チップ試作を目指すにあたり，FPGA における動作検証は不可欠である．また，FPGA を用いることで，メモリまで含めたテストが容易になる．従って，本節では提案アーキテクチャの FPGA 実装方法を示す．

図 6.7 に暗号システム全体構成を示す．図 1.1 における最下層部分 (Galois Field の演算) を Galois Field Core 部が担当しそれ以上の層は CPU で実行する．OpenCores で推奨されているバス規格 WISHBONE[29] で CPU と Galois Field Core 部を接続する．CPU には Lattice Semiconductor 社が提供しているオープンソースの CPU コア LatticeMico32[15] を利用した．また，SRAM にはターゲットである Virtex-5 のオンチップ SRAM を使用する．

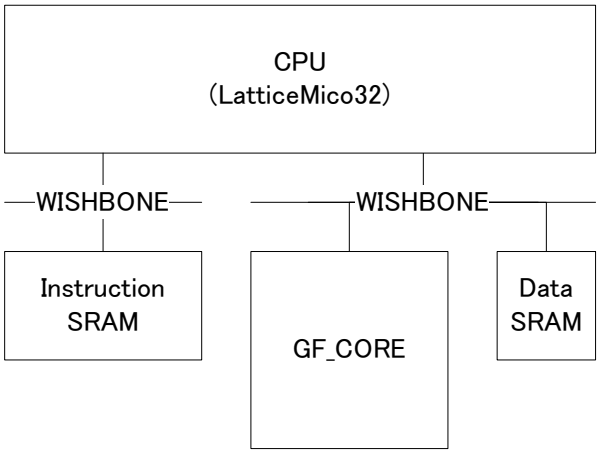

図 6.7: 暗号システム全体構成ブロック図.

LatticeMico32 は 32bit 命令長/データ長の RISC アーキテクチャの CPU コアで，WISHBONE インターフェースを持っている．LatticeMico32 を CPU として用いるメリットはデバッグのしやすいオープンソースであることと，C 言語のコンパイラが付属するなどソフトウェア開発環境が整っていることが挙げられる．また，SRAM コントローラなどの周辺回路の IP も同時にダウンロードできる．言語は

Verilog-HDL で記述されている．

図 6.8 に LatticeMico32 のブロック図を示す．LatticeMico32 はハーバードアーキテクチャを持つ．また，用途に応じて性能を選択できることも特徴である．例えば，分岐予測，外部割り込み，シフト命令，乗算除算命令，命令/データキャッシュなどを実装するかしないかを，構成ファイルを変更することで論理合成時に選択できる．今回の論理合成では，除算命令など楕円曲線暗号に必要でないものは合成対象から外した．

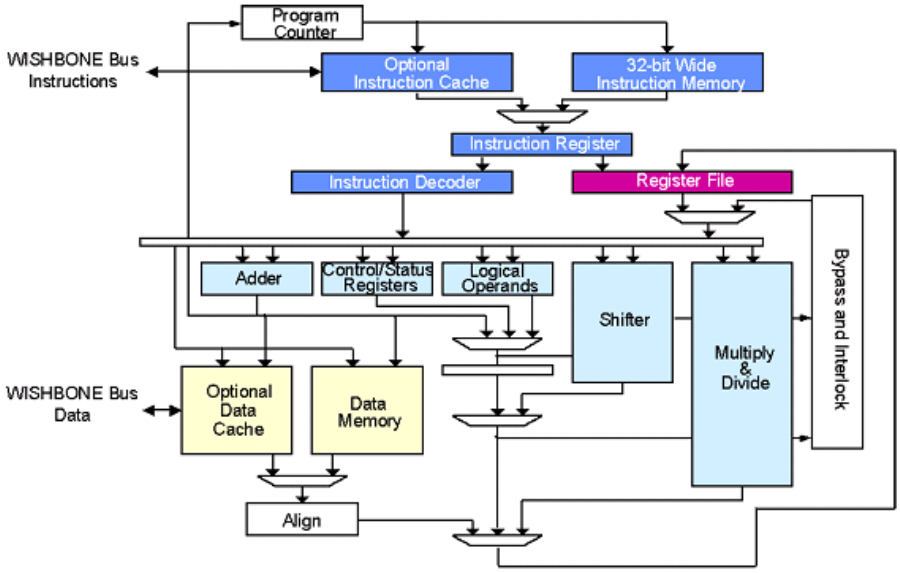

図 6.8: LatticeMico32 のブロック図.

WISHBONE は OpenCores で推奨されているバス規格である．図 6.9 に WISHBONE インターフェースを用いた標準的な接続例を示す．WISHBONE ではマスタとスレーブの 1 対 1 接続となる．

図 6.10 に Galois Field Core 部の構成を示す．I/O 部が WISHBONE とのデータの橋渡しを行う．Galois Field Core 部のコントロールは，Control Register に値を書き込むことによって行われる．Control Register の値から Controller が残りのブロックの動作をコントロールする．Parameter Registers には楕円曲線暗号に必要なパラメータを格納する．また，Data Register File には演算対象のデータと演算結果が書き込まれる．ALU は加算，減算，XOR, $GF(2^n)$ における MAC 演算,

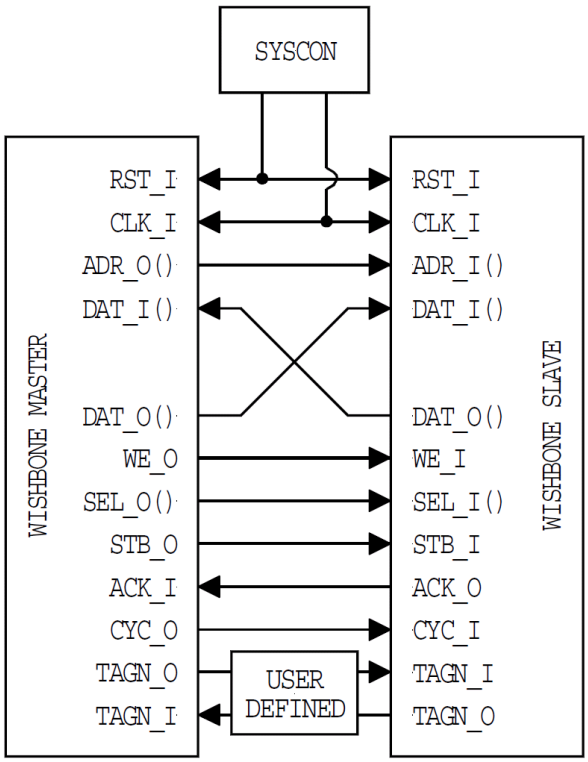

図 6.9: WISHBONE インターフェースを用いた標準的な接続例.

$GF(P)$ における MAC 演算を実行する.

図 6.11 に $GF(2^n)$ のときの MAC 演算の構成を示す．また，図 6.12 に $GF(P)$ のときの MAC 演算の構成を示す．ただし，実際には図 6.11 と図 6.12 の乗算器及びレジスターは共有され，CPU からの命令によってどちらかの構成で計算を行う.

図 6.7 に示す，Galois Field Core 部，CPU，SRAM を Verilog-HDL で記述し，Xilinx 社 Virtex-5XC5VLX220[36] をターゲットとして，同じく Xilinx 社の ISE9.1i を用いて論理合成を行った．Virtex-5XC5VLX220 では合計 34560 スライスが利用可能であり，各スライスには LUT 及びフィリップフロップが 4 個ずつ含まれる．合成対象全体のリソース使用率，及び遅延時間を評価する.

表 6.3 に Virtex-5XC5VLX220 をターゲットに論理合成した結果の比較を示す．ただし，表 6.3 中の 256 ビット MM 時間は 256 ビットモンゴメリ乗算実行時間を意味する．また，論理合成対象は図 6.7 の暗号システム全体である．クリティカ

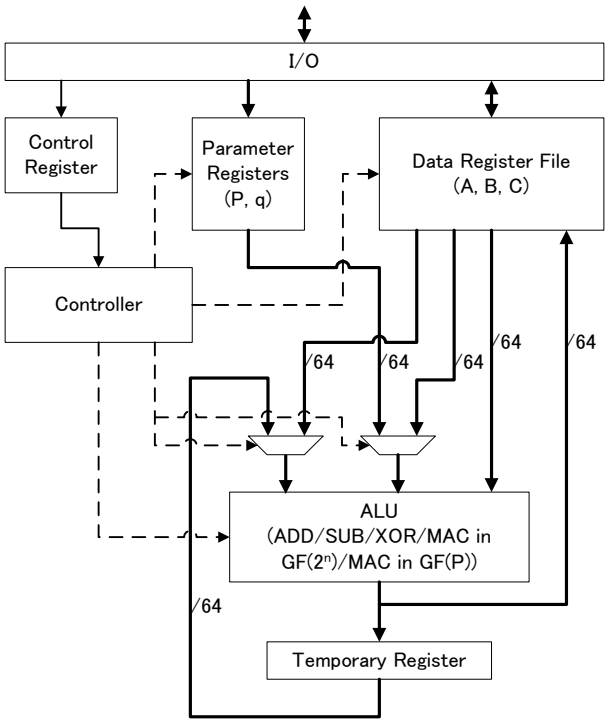

図 6.10: Galois Field Core 部の構成 (図 6.3).

表 6.2: 提案モンゴメリ乗算を Virtex-5XC5VLX220 をターゲットとして論理合成した結果

ターゲット	面積	動作周波数	ユニファイド型	フィールド	基数	256 ビット MM 時間
FPGA Virtex-5	$7712slices$	$88.2MHz$	Yes	$GF(P)$	2^{16}	$2.1\mu s$
FPGA Virtex-5	$7712slices$	$88.2MHz$	Yes	$GF(2^n)$	2^{64}	$0.63\mu s$

ルパスは提案 Galois Field Core 部内に存在する.

6.3.2　90nm プロセスライブラリを用いた論理合成結果

提案するスケーラブル双基数ユニファイド型モンゴメリ乗算器 (図 6.10) を含む ALU 部 (モンゴメリ乗算に使う積和和器，加減算器，論理積器を含む)，レジスタファイル部，コントローラ部を Verilog-HDL で記述し，Synopsys 社の Design Compiler で，STARC90nm プロセス用ライブラリを用いて論理合成を行った．次節より提案手法によるモンゴメリ乗算の演算実行時間及び，面積を比較評価する.

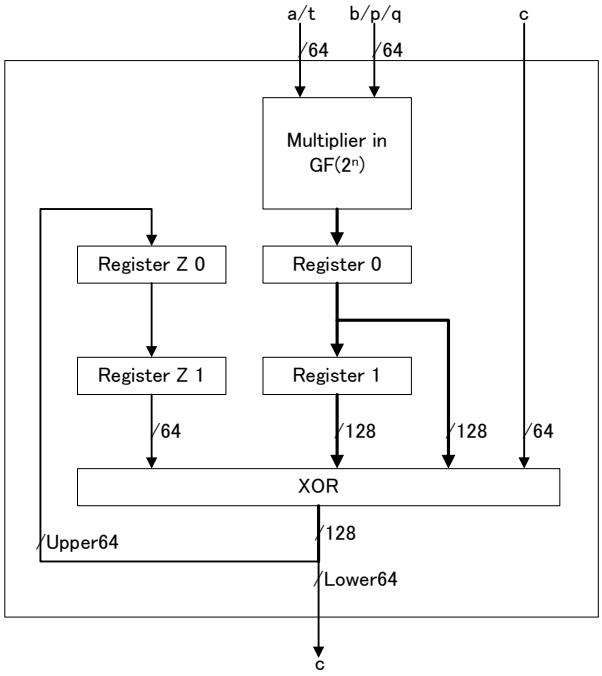

図 6.11: $GF(2^n)$ における MAC 演算時のデータパス.

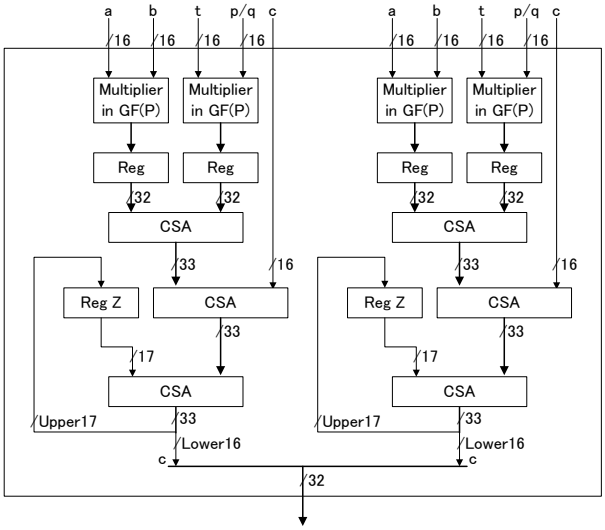

図 6.12: $GF(P)$ における MAC 演算時のデータパス.

6.4　性能評価

6.4.1　実行時間

表 6.3 に提案手法と既存手法によるモンゴメリ乗算実行時間の比較を示す．ただし，表 6.3 中の 256 ビット MM 時間は 256 ビットモンゴメリ乗算実行時間を意味する．また，提案手法の面積は NAND ゲート換算値である．ステージ数は 1 演算器を 1 ステージとし，このステージを直列に並べている数を表す．論理合成の結果では提案手法の遅延時間は $1.54ns$ であったため，理論的には $641MHz$ で動作が可能である．

表 6.3 において，提案手法による $GF(2^n)$ のモンゴメリ乗算の演算実行時間が最も短い．$GF(2^n)$ で次に高速なモンゴメリ乗算器は文献 [25] であるが，動作周波数で正規化すると，文献 [25] の手法と提案手法では 256 ビットモンゴメリ乗算実行時間は等しくなる．これは，提案手法に文献 [25] と同じアルゴリズム及び，同じ基数のアーキテクチャを用いているため，必要クロックサイクル数も等しいからである．

$GF(P)$ モンゴメリ乗算の演算実行時間では提案手法が最も短くなる．提案手法の次に $GF(P)$ モンゴメリ乗算器が高速である文献 [25] の手法に比べ，クロックサイクル数は提案手法の方が多い．しかし，$GF(2^n)$ 乗算器と同じ動作周波数で演算が可能であるため，$GF(P)$ モンゴメリ乗算全体としては提案手法は文献 [25] の手法に比べ 1.3 倍高速である．

文献 [11, 32–34] の手法ではユニファイド型ではないため，$GF(P)$ と $GF(2^n)$ の両方を扱うことが出来ない．また，文献 [9, 27] では基数が 2 であり，文献 [28] の手法では $w = 32$ としているものの，基数が $GF(P)$ のときに 2，$GF(2^n)$ のときに 2^2 と小さく，クロックサイクル数が多いため低速である．文献 [25] では，$GF(P)$ モンゴメリ演算時に比べ，$GF(2^n)$ モンゴメリ演算時にプロセッサ全体の動作周波数を大きく下げている．従って，フィールドを切り替えるために，複数の動作周波数に対応した設計にする必要がある．文献 [30] では FPGA に特化させるためビットスライス形の乗算器アーキテクチャを提案しており，遅延時間を $GF(P)$ 側

表 6.3: 提案手法と既存手法におけるモンゴメリ乗算器の性能比較

文献	テクノロジ	面積	動作周波数	ユニファイド型	フィールド	基数	ステージ数	256 ビット MM 時間
This Work	90 nm CMOS	$163kgates$[2]	$641MHz$	Yes	$GF(P)$	2^{16}	1(4parallelized)	$0.28\mu s$
This Work	90 nm CMOS	$163kgates$[2]	$641MHz$	Yes	$GF(2^n)$	2^{64}	1	$86ns$
[9]	Xilinx Virtex Pro	$1514LUTs$	$144MHz$	Yes	$GF(P)$	2	64	$1.1ms$
[9]	Xilinx Virtex Pro	$1514LUTs$	$144MHz$	Yes	$GF(2^n)$	2	64	-
[11]	Xilinx Virtex II	$2593LUTs$	$135MHz$	No	$GF(P)$	2^{16}	16	$200\mu s$
[25]	$0.13\mu m$CMOS	$107kgates$	$137.7MHz$	Yes	$GF(P)$	2^{64}	1	$0.48\mu s$
[25]	$0.13\mu m$CMOS	$107kgates$	$510.2MHz$	Yes	$GF(2^n)$	2^{64}	1	$98ns$
[27]	$1.2\mu m$CMOS	–	$80MHz$	Yes	$GF(P)$	2	2	$6.6\mu s$
[27]	$1.2\mu m$CMOS	–	–	Yes	$GF(2^n)$	2	2	–
[28]	$0.5\mu m$CMOS	$30kgates$	$203MHz$	Yes	$GF(P)$	2	2	$2.6\mu s$
[28]	$0.5\mu m$CMOS	$30kgates$	–	Yes	$GF(2^n)$	2^2	2	–
[30]	Xilinx VirtexII Pro	$5892LUTs$	$140MHz$	Yes	$GF(P)$	2^{64}	1	$7.3\mu s$
[30]	Xilinx VirtexII Pro	$5892LUTs$	$140MHz$	Yes	$GF(2^n)$	2^{64}	1	$7.3\mu s$
[32]	$0.5\mu m$CMOS	$28kgates$	$80MHz$	No	$GF(P)$	2	40	$7.4\mu s$
[34]	$0.5\mu m$CMOS	$28kgates$	$64MHz$	No	$GF(P)$	2^3	15	$3.2\mu s$

で合わせているため，$GF(2^n)$ 側で性能を引き出せていない．

6.4.2　面積

表 6.3 において，FPGA による実装以外の手法と比較して提案手法の面積が一番大きくなっている．これは，提案手法の面積に図 6.3 におけるレジスタファイル部 $124kgates$ を含んでいるからである．対して，文献 [27, 28, 32–34] に含まれる面積は，演算器のみで，これらの手法でもオペランドと演算結果を保持する外部 RAM が必要である．また，文献 [25] に含まれる面積は演算器とコントローラ部である．提案手法のコントローラ部の面積は $3kgates$，ALU 部の面積は $31kgates$ で，合計 $39kgates$ であった．従って，提案手法のコントローラ部と ALU 部の面積は文献 [27, 28, 32–34] とほぼ同じで，文献 [25] より小さい．これは $GF(P)$ の 16bits 乗算器 4 個と $GF(2^n)$ の 64bits 乗算器の共通部分を統合しているためである．表 6.4 に提案乗算器の面積の内訳を示す．

[2] 4096 ビットのレジスタファイルの面積を含む．他の手法は含まない．

表 6.4: 提案乗算器の面積の内訳.

機能ブロック	面積 ($kgates$)
ALU	31
Controller	3
Register File	124
Other	5
Total	163

6.5　本章のまとめ

本章では，$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブル双基数ユニファイド型モンゴメリ乗算器アーキテクチャを提案した．これを FPGA 上で実装検証を行い，その結果について示した．また，90nm プロセスライブラリを用いた論理合成による，面積と遅延時間の見積もり結果を示した．この論理合成結果に基づき，提案乗算器の遅延時間及び面積の評価を行い，既存手法との比較考察を行った．

6.2 節「**提案モンゴメリ乗算器アーキテクチャ**」では，$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブル双基数ユニファイド型モンゴメリ乗算器アーキテクチャを提案した．

6.3 節「HDL **による実装**」では，6.2 節にて提案したモンゴメリ乗算器を HDL により実装し，FPGA 上での実装検証及び 90nm プロセスライブラリを用いた論理合成結果結果について示した．

6.4 節「**性能評価**」では，6.3 節の論理合成結果に基づき，提案乗算器の遅延時間及び面積の評価を行い，既存手法との比較考察を行った．

第7章

結論

本論文では，公開鍵暗号の説明と，楕円曲線暗号の位置付けについて述べた．また，楕円曲線暗号のアルゴリズムを示し，実装に必要な有限体上の演算，楕円曲線の式，加法群の定義，座標系の選択，スカラー乗算について示した．その上で，モンゴメリ乗算アルゴリズムと既存モンゴメリ乗算器の問題点を示した．この研究背景から，$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブル双基数ユニファイド型モンゴメリ乗算器を提案した．各章の内容を以下にまとめる．

第 2 章「**公開鍵暗号**」では，楕円曲線暗号を含む公開鍵暗号方式について示した．公開鍵暗号の安全性は離散対数問題の困難さに基づいており，楕円曲線暗号は鍵長が小さくても RSA 暗号と同等の安全性を持つ理由を述べた．そして，リソースが限られている用途に楕円曲線暗号が有利である理由を示した．

第 3 章「**楕円曲線暗号アルゴリズム**」では，楕円曲線暗号を実装するにあたり必要なアルゴリズムについて示した．有限体の演算は多くの手法がすでに提案されており，各座標系における楕円曲線上の点の演算及び，楕円曲線上のスカラー乗算について示した．

第 4 章「**モンゴメリ乗算アルゴリズム**」では，楕円曲線暗号の中で最も支配的な演算となる剰余乗算を，効率よく実行するモンゴメリ乗算のアルゴリズムを示した．最初に $GF(P)$ 及び $GF(2^n)$ におけるモンゴメリ乗算アルゴリズムを示し，次に，これらをスケーラブルに計算できるモンゴメリ乗算アルゴリズムを示した．

第 5 章「**既存モンゴメリ乗算器**」では，既存のモンゴメリ乗算器について示した．それぞれの既存モンゴメリ乗算器のアーキテクチャの分類を行い，各々の長所及び短所を示した．その上で，スケーラブル双基数ユニファイド型モンゴメリ乗算器アーキテクチャの必要性を示した．

第 6 章「**提案モンゴメリ乗算器**」では，$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブル双基数ユニファイド型モンゴメリ乗算器アーキテクチャを提案した．これを FPGA 上で実装検証を行い，その結果について示した．また，90nm プロセスライブラリを用いた論理合成による，面積と遅延時間の見積もり結果を示した．この論理合成結果に基づき，提案乗算器の遅延時間及び面積の評価を行い，既存手法との比較考察を行った．提案モンゴメリ乗算器は，既存手法より $GF(P)$ が 30%高速で，$GF(2^n)$ 及び $GF(P)$ における両乗算を同じ周波数で動作可能であることを示した．

本論文で提案した $GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブルモンゴメリ乗算器を用いることによって，様々なセキュリティレベルの公開鍵暗号を，より高速に暗号化及び復号することが可能となる．

今後の課題としては，チップ試作に向け，ゲートレベルでの動作検証や配置・配線が挙げられる．

謝辞

本論文を執筆するにあたり数々の貴重な御指導，御助言を賜りました本学理工学術院基幹理工学研究科，柳澤政生教授に深く感謝致します.

本論文全般にわたり様々な御指導，御助言を頂きました本学理工学術院基幹理工学研究科，大附辰夫教授及び戸川望准教授に深く御礼申しあげます.

最後に，日頃より多方面にわたり様々な御意見，御助言を頂きました大附・柳澤研究室の皆様に心より感謝致します.

参考文献

[1] T. Acar, "High-speed algorithms and architectures for number-theoretic cryptosystems," PhD thesis, Oregon State University, 1998.

[2] E. Al-Daoud, R. Mahamod, M. Rushadan, and A. Kilicman, "A new addition formula for elliptic curves over $GF(2^n)$," *IEEE Transactions on Computers,* Vol. 51, No. 8, pp.972-975, August 2002.

[3] S. Bartolini, I. Branovic, R. Giorgi, and E. Martinelli, "A Performance Evaluation of ARM ISA Extension for Elliptic Curve Cryptography over Binary Finite Fields," *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing,* pp.238-245, October 2004.

[4] S. Bartolini, G. Castagnini, and E. Martinelli, "Inclusion of a Montgomery Multiplier Unit into an Embedded Processor's Datapath to Speed-up Elliptic Curve Cryptography," *Proceedings of the Third International Symposium on Information Assurance and Security,* pp.95-100, August 2007.

[5] "Digital Signature Standard (DSS)," FIPS PUB 186-2, National Institute of Standards and Technology, `http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf`, October 2001.

[6] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory,* Vol. 31, Issue 4, pp.469-472, July 1985.

[7] J. Goodman and A. Chandrakasan, "An Energy Efficient Reconfigurable Public-Key Cryptography Processor Architecture," *Proceedings of the Sec-*

ond International Workshop Cryptographic Hardware and Embedded Systems, pp.175-190, August 2000.

[8] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography,* Springer-Verlag, July 2003.

[9] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu, "An Improved Unified Scalable Radix-2 Montgomery Multiplier," *Proceedings of the 17th IEEE Symposium on Computer Arithmetic,* June 2005.

[10] "IEEE Std 1363-2000, IEEE Standard Specifications for Public-Key Cryptography," `http://ieeexplore.ieee.org/iel5/7168/19282/00891000.pdf,` January 2000.

[11] K. Kelley and D. Harris "Parallelized Very High Radix Scalable Montgomery Multipliers," *Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers,* pp.1196-1200, October 2005.

[12] D. Knuth, *The Art of Computer Programming–Seminumerical Algorithms,* Addison-Wesley, 3rd edition, 1998.

[13] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation,* Vol. 48, pp.279-287, 1987.

[14] Ç.K. Koç, T. Acar, and B.S. Kaliski, "Analyzing and Comparing Montgomery Multiplication Algorithms," *IEEE Micro,* Vol.16, No.3, pp.26-33, June 1996.

[15] "LatticeMico32 Open, Free 32-Bit Soft Processor," Lattice Semiconductor Corporation, `http://www.latticesemi.com/products/intellectualproperty/ipcores/mico32/index.cfm,` 2007.

[16] J. López and R. Dahab, "Improved Algorityms for elliptic curve arithmetic in $GF(2^n)$," *Selected aread in cryptography,* Springer-Verlag, Lecture Notes in Computer Science, 1556, pp.201-212, August, 1998.

[17] J. López and Ricardo Dahab, "Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation," *Cryptographic Hardware and Embedded Systems,* Springer-Verlag, 1717, Lecture Notes in Computer Science, pp.316-327, August 1999.

[18] C.J. Mcivor, M. Mcloone, and J.V. Mccanny, "Hardware Elliptic Curve Cryptographic Processor Over $GF(p)$," *IEEE Transactions on Circuits and Systems I,* Vol.53, Issue 9, pp.1946-1957, Septmber 2006.

[19] V. Miller, "Uses of elliptic curves in cryptography," *Advances in Cryptology,* Springer-Verlag, Lecture Notes in Computer Science, 218, pp.417-426, 1986.

[20] P.L. Montgomery, "Modular Multiplication without Trial Division," *Math. Computing,* Vol.44, No.170, pp.519-521, April 1985.

[21] 岡本　龍明, 内山　成憲, "楕円曲線の安全性について," 情報処理, 情報処理学会, Vol. 39, No. 12, pp.1252-1257, December 1998.

[22] K. Okeya, "Efficient elliptic curve cryptosystems from a scalar multiplication algorithhm with recovery of the y-coordinate on a Montgomery-form elliptic curve," *Cryptographic Hardware and Embedded Systems,* Springer-Verlag, Lecture Notes in Computer Science, 2162, pp.126-141, 2001.

[23] G. Orlando and C. Paar, "A Scalable $GF(p)$ Elliptic Curve Processor Architecture for Programmable Hardware," *Proceedings of the Third International Workshop Cryptographic Hardware and Embedded Systems,* pp.349-363, May 2001.

[24] R. Rivest, A. Shamir, and L. Andleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM,* Vol. 21, pp.120-126, 1978.

[25] A. Satoh and K. Takano, "A Scalable Dual-Field Elliptic Curve Cryptographic Processor," *IEEE Transactions on Computers,* Vol.52, No.4, pp.449-460, April 2003.

[26] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "Multicore Curve-Based Cryptoprocessor with Reconfigurable Modular Arithmetic Logic Units over $GF(2^n)$," *IEEE Transactions on Computers,* Vol.56, No.9, pp.1269-1282, September 2007.

[27] E. Savas, A.F. Tenca, and Ç.K. Koç, "A Scalable and Unified Multiplier Architecture for Finite Fields $GF(p)$ and $GF(2^m)$," *Proceedings of the Second International Workshop Cryptographic Hardware and Embedded Systems,* pp.277-292, August 2000.

[28] E. Savas, A.F. Tenca, and Ç.K. Koç, "Multiplier Architectures for $GF(p)$ and $GF(2^n)$," *Proceedings of IEE Computers and Digital Techniques,* Vol.151, No.2, pp.147-160, March 2004.

[29] "SoC Interconnection: Wishbone," OPENCORES.ORG, `http://www.opencores.org.uk/projects.cgi/web/wishbone/`, July 2002.

[30] M. Sudhakar, R.V. Kamala, and M.B. Srinivas, "A bit-sliced, scalable and unified montgomery multiplier architecture for RSA and ECC," *Proceedings of IFIP International Conference on Very Large Scale Integration,* pp.252-257, October 2007.

[31] K. Tanimura, R. Nara, S. Kohara, K. Shimizu, Y. Shi, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Scalable Unified Dual-Radix Architecture for Montgomery Multiplication in $GF(P)$ and $GF(2^n)$," *Proceedings of the 13th IEEE Asia and South Pacific Design Automation Conference 2008,* pp.697-702, January 2008.

[32] A.F. Tenca and Ç.K. Koç, "A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm," *IEEE Transactions on Computers,* Vol.52, No.9, pp.1215-1221, September 2003.

[33] A.F. Tenca and Ç.K. Koç, "A Scalable Architecture for Montgomery Multiplication," *Proceedings of the First International Workshop Cryptographic Hardware and Embedded Systems,* pp.94-108, August 1999.

[34] A.F. Tenca, G. Todorov, and Ç.K. Koç, "High-Radix Design of a Scalable Modular Multiplier," *Proceedings of the Third International Workshop Cryptographic Hardware and Embedded Systems,* pp.185-201, May 2001.

[35] J. Uchida, N. Togawa, M. Yanagisawa and T. Ohtsuki, "A Fast Elliptic Curve Cryptosystem LSI Embedding Word-Based Montgomery Multiplier," *IEICE Transactions on Electronics,* Vol.E89-C, No.3, pp.243-249, March 2006.

[36] Xilinx, Inc, `http://www.xilinx.com/`, January 2008.

2007年度 修士論文

$GF(2^n)$及び$GF(P)$における楕円曲線暗号向けスケーラブルモンゴメリ乗算器に関する研究

指導教授　柳澤 政生　教授

早稲田大学大学院 理工学研究科
情報・ネットワーク専攻

3606U120–9

谷村 和幸

Kazuyuki Tanimura

二〇〇七年度　修士論文 $GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブルモンゴメリ乗算器に関する研究　谷村 和幸

二〇〇七年度　修士論文 $GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブルモンゴメリ乗算器に関する研究　谷村 和幸

二〇〇七年度　修士論文 $GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブルモンゴメリ乗算器に関する研究 谷村 和幸

二〇〇七年度　修士論文　$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブルモンゴメリ乗算器に関する研究　谷村 和幸

二〇〇七年度　修士論文　$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブルモンゴメリ乗算器に関する研究　谷村 和幸

二〇〇七年度　修士論文　$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブルモンゴメリ乗算器に関する研究　谷村 和幸

二〇〇七年度　修士論文　$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブルモンゴメリ乗算器に関する研究　谷村 和幸

二〇〇七年度　修士論文　$GF(2^n)$ 及び $GF(P)$ における楕円曲線暗号向けスケーラブルモンゴメリ乗算器に関する研究　谷村 和幸