公立はこだて未来大学 2024 年度 システム情報科学実習
グループ報告書

Future University Hakodate 2024 Systems Information Science Practice
Group Report

プロジェクト名

暗号と数理とセキュリティ

Project Name

Cryptography, Mathematics, and Security

プロジェクト番号/Project No.

4

プロジェクトリーダ/Project Leader

後藤瞭介　Ryousuke Goto

プロジェクト/Project Member

後藤瞭介　Ryousuke Goto
ディオガーディディラン基暉　DylanMotoki Dioguardi
伊澤叡遵　Eijun Izawa
松浦実里　Minori Matsuura

指導教員

白勢政明　川越敏司　川口聡　齊藤朝輝

Advisor

Masaaki Shirase　Toshiji Kawagoe　Satoshi Kawaguchi　Asaki Saito

提出日

2024 年 1 月 21 日

Date of Submission

January 21, 2024

目次

第 1 章　はじめに

1.1　背景

現代社会において、情報通信技術の発展は目覚ましいものがあり、それに伴いネットワーク社会の形態も急速に進化している。インターネットを介した情報のやり取りが日常生活の一部となり、個人や企業にとってネットワークの利用は不可欠なものとなっている。しかし、このような利便性の裏には、セキュリティに関する多くの課題が存在する。データ漏洩やサイバー攻撃のリスクは年々増加しており、セキュリティ対策の重要性が一層高まっている。セキュリティの重要性については広く認識されているものの、その具体的なシステムやコストに関する理解は一般的には十分とは言えないだろう。例えば、ブロックチェーン技術に関連する話題としてビットコインが有名だが、その裏で国一つ分に匹敵する電力がただの計算処理のために消費されていることを知っている人は少ないだろう。このような技術の持つ社会的課題に対する認識を深め、解決策を模索することが求められているといえるのではないだろうか。本プロジェクトでは、公開鍵暗号方式やブロックチェーンなど世の中にある暗号技術について学習し、それらに関連のある社会に存在する課題を踏まえ、解決することを目的とした。

1.2　課題の概要

今回取り上げる解決したい課題は一つは、楕円曲線暗号のアルゴリズムをハードウェアに FPGA（Field Programmable Gate Array）を用いて実装することである。FPGA とは、ユーザーが自ら設計し、プログラム可能な集積回路の一種である。利点としては、本来暗号化処理を行うはずの CPU 負担を軽減することができる。また、高位合成を用いることで、プログラミング言語により FPGA 上に回路を設計することができる。

（※文責: 後藤瞭介）

第 2 章　関連研究

2.1　前年度プロジェクトにおける目的

前年度プロジェクトでは、二つの目的を掲げてプロジェクト学習を遂行した。

一つ目の目的は、加速する AI 事情および量子コンピューターが実用化に向けられていることによって暗号技術の安全性が脅かされていることに伴い、次世代の暗号技術である格子暗号の開発に挑戦することであった。具体的には、格子暗号の暗号方式である LWE 方式、GGH 方式において暗号化と復号を行えるプログラムを作成し、Web 上に搭載することを最終目標としていた。

二つ目の目的は、現在世界中で普及している楕円曲線暗号を使い、公立はこだて未来大学の学生が身近に利用できるブロックチェーン技術を実装することであった。具体的には、ブロックチェーン技術とそれに関わるマイニングなどの作業を、人々が手軽に体験できる Web アプリケーションの開発を最終目標としていた。

2.2　グループ A（格子暗号の開発）

2.2.1　必要なスキル

格子暗号の開発を行うにあたって必要なスキルは、主に三つある。

一つ目は、格子暗号の基礎理解に必要な数論の知識を習得するために、努力できるスキルである。暗号の仕組みや安全性について理解するためには、数論の知識が重要である。格子暗号の場合は、格子や格子基底、ユニモジュラ行列などの知識を習得することが重要であるため、これらの知識を習得するために努力できるスキルが求められる。

二つ目は、Python でプログラムを作成するためのプログラミングスキルである。このグループでは、格子暗号のプログラムを Python で作成していた。そのため、Python の言語を学び、Python でプログラムを作成できるスキルを身に着けることが求められる。

三つ目は、Web フレームワーク Flask を扱うスキルである。このグループでは、Python で作成した暗号プログラムを、Flask を用いて Web 上に搭載していた。そのため、事前に Flask について学習し、Flask を扱うためのスキルを身に着けることが求められる。

2.2.2　解決手法

前期（4〜7 月）は、格子暗号の基礎について学ぶために、離散数学の教科書を用いて輪読を行っていた。そして後期（8〜12 月）は、夏休みの間に、後期で使用する Web フレームワーク Flask の学習を行い、格子暗号のプログラムを Python で作成していた。その後、作成したプログラムを、10 月から 11 月にかけて Flask を用いて Web 上に搭載することを行っていた。

2.3 グループ B（ブロックチェーン技術の実装）

2.3.1 必要なスキル

ブロックチェーン技術の実装に必要なスキルは、主に三つある。

一つ目は、楕円曲線暗号の基礎を理解するために必要な楕円曲線の知識と有限体の知識を習得するために、努力できるスキルである。楕円曲線暗号の詳細な仕組みを理解するためには、有限体上の楕円曲線の知識が必須である。そのため、それらの知識を習得するために、努力できるスキルが求められる。

二つ目は、適切な情報を収集するスキルである。このグループでは、ブロックチェーンは比較的新しい技術であるため、日本語だけでなく英語での情報収集も必要であると考えていた。そのため、様々な情報源を閲覧する必要があるので、その中から適切な情報を選び取り、収集するスキルはとても重要である。

三つ目は、Python でプログラムを作成するためのプログラミングスキルである。このグループでは、Python でプログラミングされた BitCoin において、楕円曲線暗号がどのようなコードで記述されているか学んでいた。そして学んだ後、楕円曲線暗号を用いたデジタル署名を、既存のライブラリなしで Python で実装する方法を模索していた。これらの過程で、Python に関する基本的な知識を習得していることと、Python でプログラミングを作成できるスキルが求められる。

2.3.2 解決手法

手始めに、楕円曲線暗号の基礎を理解するために、公立はこだて未来大学で 1 年生後期に開講される情報数学の教科書と、担当教員の推薦に基づく楕円曲線に関する基礎の本をもとに、有限体と楕円曲線について学習していた。そして次に、楕円曲線暗号や BitCoin、ブロックチェーン技術について、情報ライブラリの書籍とインターネットを活用して情報を収集していた。そして製作物のアイデアを選定した後、ブロックチェーンや暗号資産についてさらなる調査を行い、Python の既存のライブラリを利用したブロックチェーンの実装と、既存のライブラリを利用しない楕円曲線暗号を用いたデジタル署名の実装を行っていた。

（※文責: 後藤瞭介）

第 3 章　プロジェクト学習の目標

本プロジェクトは、デジタル社会における情報セキュリティの課題を解決し、より安全で信頼性の高い情報社会の構築に貢献することを目的とする。情報セキュリティは、現代のデジタル社会において重要な要素であり、暗号技術はその根幹を支える基本技術の一つである。特に、データの保護、認証、プライバシーの確保といった分野において、暗号技術は不可欠な存在であり、その理解と応用は、社会のあらゆる分野での安全性向上に資する。

具体的に前期では、暗号技術の基礎である数理の学習を行うことを主眼とする。これには、数論、代数、離散数学などの基礎的な数理知識の修得が含まれ、これらの知識をもとに暗号アルゴリズムの動作原理を理解することを目指す。また、情報セキュリティの理論的裏付けを深めることで、今後の応用研究に必要な基礎を確立することが期待される。

後期は、現代社会における実用的な課題として、自動運転技術や IoT デバイスなどの分野に着目する。これらの分野では、膨大なデータがリアルタイムで処理される必要があり、暗号処理の高速化が重要な課題となっている。特に、自動運転車においては、通信データの暗号化と復号を低遅延で行うことが求められ、安全性と効率性の両立が必須である。このような背景を踏まえ、ハードウェアによる暗号処理の最適化が重要な研究テーマだと考えていた。そこで、前期で得た数理的基礎を活用し、後期において楕円曲線暗号のハードウェア実装を行うことを目指す。楕円曲線暗号は、従来の公開鍵暗号方式と比較して、同等の安全性をより短い鍵長で実現できる点で注目されている。また、FPGA（Field-Programmable Gate Array）を用いたハードウェア実装は、柔軟性と性能の両面で利点があり、暗号技術の高速処理において効果的な手法である。加えて、設計フローの実践を通じて、効率的なハードウェア開発プロセスを追求することは、ソフトウェアとハードウェアの設計プロセスにおける共通点や相違点を理解し、ハードウェア実装における課題を特定することにつながる。また、暗号技術の実用化に向けたハードウェア最適化の可能性を探り、情報セキュリティ分野における研究の発展に寄与され、デジタル社会のさらなる安全性向上に向けた貢献を果たすことにつながると考えた。

そのため、ソフトウェア開発における設計フローを活かしながら、高位合成を通して、ハードウェア実装に取り組むことで、その有用性を確認して、記録しておく。さらに、実装したものとソフトウェアベースの暗号処理との性能比較を通じて、ハードウェア実装の有効性を検証する。これにより、実用的な暗号技術の社会実装に向けた知見を深めることを最終的な目的とした。

（※文責: 後藤瞭介）

第 4 章　目的を達成するための手法、手段

4.1　学習内容

4.1.1　数論の学習

楕円曲線は現代社会の暗号技術において、多岐にわたる分野で利用されている。具体的には、ブロックチェーンの台帳技術や Web 通信技術における重要な証明書である TLS1.3、時期マイナンバーカードなどに応用されている。本プロジェクトでは、現代社会で広く普及している公開鍵暗号技術である楕円曲線暗号およびその応用技術に焦点を当てることとした。

そのため、前期においては楕円曲線の基礎理論の理解を目的とし、Joseph H. Silverman 氏の著書『はじめての数論』第 4 版を基に学習を進めた。特に公開鍵暗号において必要不可欠な整数論の知識に重点を置きながら、楕円曲線に関連する数理の理解を深めた。

以下は本書で学んだ暗号に関連する内容である。

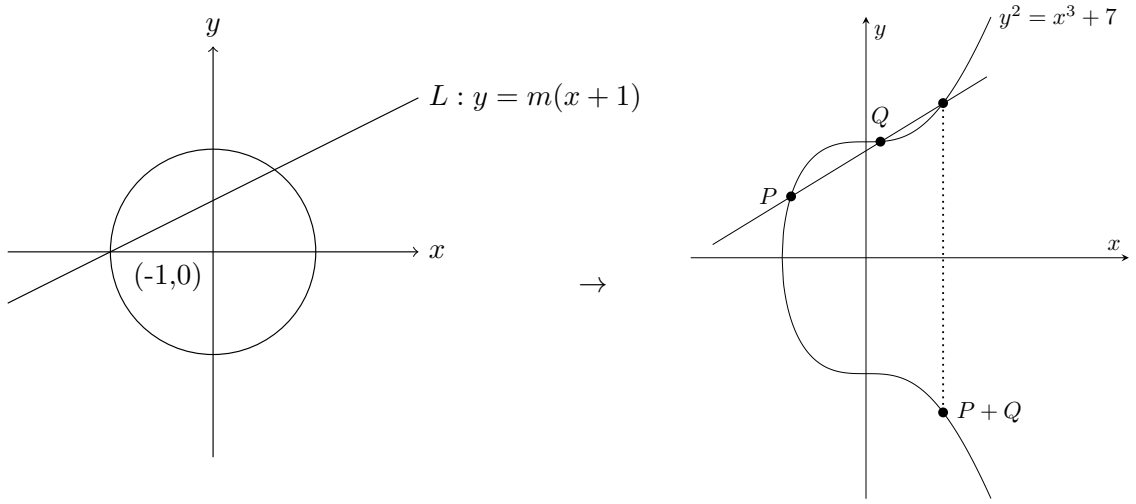


図 4.1　円と楕円曲線のグラフ

数論的な考え方を身につけるために、円周上の有理点を求めることから始めた。この考え方を応用することで、楕円曲線上の点 P と Q に対して、P+Q を定めることが出来る。この性質は、楕円曲線暗号の計算において重要な役割を果たしている。また、RSA 暗号や楕円曲線暗号には剰余の概念が深く関わっている。従って、合同式の復習を行い、剰余の世界での計算方法について学んだ。また、剰余の概念を応用し、中国の剰余定理についても学んだ。この定理は RSA 暗号の計算において重要な役割を果たしている。その後、白勢先生による講話で実際にこれらの数理がどのように暗号技術で用いられているのか理解を深めた。

（※文責: 伊澤叡遵）

4.1.2　公開鍵暗号の学習

担当教員の元、公開鍵暗号について学習をした。

図 4.2　公開鍵暗号のしくみ

1.A さんは、B さんの公開鍵を使って平文 m を暗号化し、B さんに送る。

2.B さんは、暗号化された m を受信し、自身の秘密鍵で復号する。

楕円曲線暗号 (ECC) は公開鍵暗号の 1 つであり、楕円曲線上の有理点の加算を用いて暗号化と復号を行うものである。ECC は、RSA 暗号に比べてより短い鍵長で同等の暗号強度を実現できるため、高速処理が可能である。ECC の安全性の根拠は、公開鍵から秘密鍵を求める計算に膨大な時間がかかることにある。

（※文責: 伊澤叡遵）

4.1.3　FPGA

ハードウェア実装において、仮想通貨のマイニングなどに使われている ACIS のような専用デバイスの製造や従来の回路設計はコストが高くなることが課題であった。そこで、本プロジェクトでは FPGA（Field Programmable Gate Array）というデバイスを活用した。FPGA とは、プログラム可能な論理回路を集積したデバイスであり、ユーザーが設計情報を外部から書き込むことで、特定の用途に合わせたカスタマイズが可能である。これを利用することで、高コストな専用回路を製造することなく、プロトタイピングや多用途向けの開発が効率的に行える。特に、高速処理や設計の柔軟性が求められる場面で重要な役割を果たしている。FPGA の活用により、コストを抑えつつ、開発スピードを向上させることが可能である。

（※文責: 伊澤叡遵）

4.1.4　高位合成（High-Level Synthesis）

高位合成とは、C 言語などのソフトウェア言語で記述されたコードを元に、ハードウェア記述言語（HDL）に自動変換することである。従来、FPGA の設計には HDL が用いられるが、これには高度な専門知識と多くの時間が必要である。そこで、設計の効率化を図るために高位合成が活用した。高位合成を用いることで、学生主体でも比較的扱いやすい C 言語で回路設計が可能で、複雑なハードウェア構造の設計やテストがより容易になる。具体的には、高位合成ツールが C 言語で記述されたコードを解析し、FPGA で実行可能な論理回路として自動的に生成する。また、高位合成は、ハードウェア設計とソフトウェア開発の融合を促進する技術として注目されており、特に AI やデータ処理、通信システムなど、多様な応用分野で役立っている。

（※文責: 伊澤叡遵）

4.1.5　高位合成ソフトの学習

ハードウェア実装には AMD(旧:Xilinx) の FPGA を利用した。従って、AMD が提供しているソフトウェアで高位合成を行う。今回のプロジェクトでは 2023 年に新しく登場した Vitis 統合 IDE を主に学習した。Vitis 統合 IDE は既存の Vitis HLS と Vitis IDE (Vitis Classic) を融合したものであり、Xilinx が今後サポートし続ける高位合成ツールである。Vitis 統合 IDE の学習は AMD が提供するユーザーガイドやリファレンスマニュアルをもとに行った [4][5]。また学習の一貫として、今後本プロジェクトを発展することがあれば、それが容易になるようハードウェア実装の方法を記録に残した。詳細はハードウェア実装の手法の章に記載している。

（※文責: ディオガーディディラン基暉）

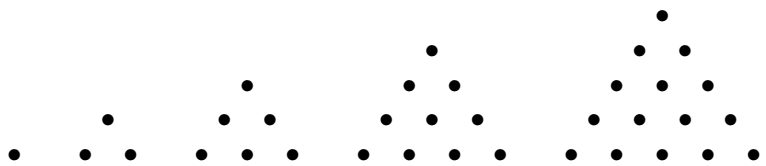
この節でのサブセクションは書籍の章タイトルである。

4.1.6　数論とは何ものでしょう？

この章では、数論の基礎となる様々な種類の数の定義と、特殊な形の数について書かれていた。

まず、自然数、整数、有理数、実数、複素数の定義とそれぞれの特徴について学習した。特に、自然数が数論の中心的な研究対象であることを理解した。

次に、平方数と三角数について深く学んだ。平方数は n^2 の形で表される数で、1, 4, 9, 16, 25, ... という数列を形成する。三角数は $\frac{n(n+1)}{2}$ で表される数で、1, 3, 6, 10, 15, ... という数列になる。これらの数の性質や相互関係について探究した。



特に興味深かったのは、平方数と三角数の両方になる数の探索である。例えば、1 と 36 は両方の性質を持つことを発見し、このような数がほかにも存在するかどうかを調べた。さらに、N^2+1 の形の素数についても調べた。この形の数が素数になるかどうかは、数論における興味深い問題の一つである。

例えば、

- $2^2+1=5$ (素数)
- $4^2+1=17$ (素数)
- $6^2+1=37$ (素数)

などの例を見つけた。

しかし、すべての N^2+1 が素数になるわけではないことも確認した（例：$5^2+1=26$ は合成数)。この節の学習を通じて、単純な数の概念から出発して、複雑で深遠な問題へと発展していく数論の魅力を実感した。また、数のパターンを観察し、仮説を立て、それを検証するという数学的思考プロセスの重要性も学ぶことができた。

4.1.7　三平方の定理とピタゴラス数

この節では、三平方の定理とピタゴラス数について深く学んだ。ピタゴラス数とは、方程式 $a^2 + b^2 = c^2$ を満たす自然数の組 (a, b, c) のことを指す。

特に重要なのは、既約ピタゴラス数の概念である。これは、a, b, c の最大公約数が 1 であるピタゴラス数のことを指す。既約ピタゴラス数は、すべてのピタゴラス数の基本的な構成要素となっている。

この節の中心的な成果は、既約ピタゴラス数の一般形の導出である。任意の互いに素な奇数 $s > t$ に対して、

$$
\begin{aligned}
a &= st \\
b &= \frac{s^2 - t^2}{2} \\
c &= \frac{s^2 + t^2}{2}
\end{aligned}
$$

という形で既約ピタゴラス数を生成できることを学んだ。

さらに、ピタゴラス数の幾何学的意味についても学んだ。各ピタゴラス数は、整数の辺の長さを持つ直角三角形に対応している。

最後に、ピタゴラス数が無限に存在することの証明を学んだ。この証明は、数学的帰納法や無限降下法といった証明技法の応用例として重要である。

4.1.8　ピタゴラス数と円周上の点

この節では、ピタゴラス数の概念とその一般形の導出について学んだ。

ピタゴラス数とは、方程式 $a^2 + b^2 = c^2$ を満たす整数の組 (a, b, c) のことを指す。この方程式は、直角三角形の辺の長さの関係を表す三平方の定理と密接に関連している。

特に重要なのは、既約ピタゴラス数の概念であり、a, b, c が共通因数を持たないピタゴラス数のことである。既約ピタゴラス数は、すべてのピタゴラス数の基本的な構成要素となる。

この節の中心的な成果は、既約ピタゴラス数の一般形の代数的な導出です。

$$
a = u^2 - v^2, \quad b = 2uv, \quad c = u^2 + v^2
$$

という形で既約ピタゴラス数を生成できることを証明した。

この結果の導出過程では、以下のような手順を踏んだ：

1. $a^2 + b^2 = c^2$ の両辺を c^2 で割り、$(a/c)^2 + (b/c)^2 = 1$ を得る。
2. 単位円上の有理点を考察し、$(x, y) = (\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2})$ の形で表されることを示す。
3. この表現を元の方程式に戻し、適切な変数変換を行うことで最終的な形を得る。

この導出過程を通じて、代数的手法と幾何学的直感の組み合わせが、数論的問題の解決にいかに重要であるかを理解した。

さらに、この一般形を用いることで、無限に多くの既約ピタゴラス数を生成できることも確認した。例えば：

- $s = 3$, $t = 1$ のとき、$(3, 4, 5)$

- $s = 5$, $t = 1$ のとき、$(5, 12, 13)$
- $s = 7$, $t = 1$ のとき、$(7, 24, 25)$

この章の学習を通じて、単純な数の関係から出発して複雑な理論を構築していく数学の美しさを実感した。また、代数、幾何、数論という数学の異なる分野がいかに密接に関連しているかを理解し、数学的思考の柔軟性と深さを学んだ。

4.1.9 余りを調べる–合同式

この章では、合同式 $a \equiv b \pmod{n}$ の概念と応用、特に楕円曲線暗号との関連について学んだ。

合同式は、数を法で割った余りを考える概念である。自然数 n に対して、集合 $\{0, 1, 2, \ldots, n-1\}$ を考える。この集合上で、通常の計算を行ってから n で割った余りを取ることで、加減乗算を定義できる。

特に重要なのは、法が素数 p の場合である。素数 p に対しては、集合 $\{0, 1, 2, \ldots, p-1\}$ 上で四則演算（加減乗除）が全て定義できる。これは、p が素数である時、0 以外の全ての要素が乗法の逆元を持つためである。この性質は、楕円曲線暗号を含む多くの暗号システムの基礎となっている。担当教員の説明によると、素数を法とする剰余体上での演算が暗号を学ぶ上で非常に重要であるとのことである。担当教員によると、このことは暗号を学ぶ上でとても重要である。

4.1.10 オイラーの ϕ 関数と中国の剰余定理

この章では、オイラーの ϕ 関数（オイラー関数）について学び、その暗号技術への応用を探究した。

オイラーの ϕ 関数は、正の整数 m に対して、1 から m までの整数のうち m と互いに素なものの個数を表す関数である。数学的に表現すると：

$$\phi(m) = |\{n : 1 \leq n \leq m, \gcd(n, m) = 1\}|$$

ここで、$\gcd(n, m)$ は n と m の最大公約数を表す。オイラー関数の重要な性質として、以下を学んだ：

1. m が素数 p のとき、$\phi(p) = p - 1$
2. m と n が互いに素のとき、$\phi(mn) = \phi(m)\phi(n)$
3. p が素数、k が正の整数のとき、$\phi(p^k) = p^k - p^{k-1} = p^k \left(1 - \frac{1}{p}\right)$

特に重要なのは、オイラー関数が代表的な公開鍵暗号システムの鍵生成過程で重要な役割を果たすことである。例えば、RSA 暗号では以下のようにオイラー関数が使用されている：

1. 二つの大きな素数 p と q を選び、$n = pq$ を計算する
2. $\phi(n) = \phi(p)\phi(q) = (p-1)(q-1)$ を計算する
3. 公開鍵 e を $\gcd(e, \phi(n)) = 1$ となるように選ぶ
4. 秘密鍵 d を $ed \equiv 1 \pmod{\phi(n)}$ を満たすように計算する

この過程において、$\phi(n)$ の値を知ることが鍵生成の核心部分となる。しかし、大きな合成数 n に対して $\phi(n)$ を直接計算することは、n の素因数分解と同程度に困難である。この計算の困難さが、RSA 暗号の安全性の基礎となっている。

また、オイラーの定理 $a^{\phi(m)} \equiv 1 \pmod{m}$（ただし $\gcd(a, m) = 1$）も学んだ。これは、RSA暗号の正当性の証明に使用される。

この章の学習を通じて、純粋に理論的に見える数学的概念が、現代の暗号技術において中心的な役割を果たしていることを理解した。特に、数論の深い結果が情報セキュリティの基盤となっている点に、数学の力強さと実用性を実感することができた。

4.1.11 三次曲線と楕円曲線

この章では、楕円曲線の基本的な概念と性質を記す。楕円曲線は、代数幾何学の重要な対象であり、数論においても中心的な役割を果たしている。

楕円曲線は一般に以下の形の式（ワイエルシュトラス方程式）で表される：

$$E : y^2 = x^3 + ax^2 + bx + c$$

ここで、a, b, c は定数である。特に、$E : y^2 = x^3 + ax + b$ の形（短いワイエルシュトラス方程式）がよく用いられる。

楕円曲線 E の重要な性質として、以下を学んだ：

1. 楕円曲線上の点集合に、加法演算が定義され群を形成する
2. この加法は、幾何学的には「点 P と点 Q を通る (P=Q の場合は P での E の接線) 直線が曲線と交わる第 3 の点の x 軸に対して反転」として定義される
3. 楕円曲線上の有理点の集合は有限生成アーベル群を成す（モーデル・ヴェイユの定理）
4. 楕円曲線の判別式 $\Delta = -16(4a^3 + 27b^2)$ が、曲線の幾何学的構造を決定する

楕円曲線上の点の加法について、具体的な計算方法を学んだ：

- 異なる 2 点 $P_1(x_1, y_1)$ と $P_2(x_2, y_2)$ の和 $P_1 + P_2 = P_3(x_3, y_3)$ は以下で与えられる：

$$\begin{aligned} \lambda &= \frac{y_2 - y_1}{x_2 - x_1} \\ x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned} \tag{4.1}$$

- 点 $P(x, y)$ の 2 倍 $P + P = 2P(x', y')$ は以下で与えられる：

$$\begin{aligned} \lambda &= \frac{3x^2 + a}{2y} \\ x' &= \lambda^2 - 2x \\ y' &= \lambda(x - x') - y \end{aligned} \tag{4.2}$$

すると、点 P の n 個の和

$$nP = P + P + \cdots + P$$

が定義される。P と n から nP を計算することをスカラー倍算という。

この章では、有限体、特に素数 p を法とする集合 $\{0, 1, 2, \cdots, p\}$ 上の楕円曲線について学んだ。これは楕円曲線理論の重要な応用であり、暗号技術にも直接関係している。

楕円曲線 E を素数 p を法として考えることを、$E \bmod p$ と表す。具体的には、楕円曲線の方程式

$$y^2 = x^3 + ax + b$$

の係数 a, b および座標 x, y をすべて mod$p$ で考える。$\{0, 1, 2, \cdots, p\}$ では四則演算ができるため、mod$p$ で式 (3.1) と (3.2) を使って $P + Q$ や $2P$ を計算でき、更に nP を計算できる。

この章で学んだ主要な概念と結果は以下の通りである：

1. $E \bmod p$ 上の点の数 N_p：
   - N_p は、方程式 $y^2 \equiv x^3 + ax + b \pmod{p}$ を満たす (x, y) の組の数
   - N_p の値は、p や楕円曲線の係数に依存して変化する
2. Hasse の定理：N_p は以下の不等式を満たす

$$|p - N_p| \leq 2\sqrt{p}$$

3. p-欠損 a_p：

$$a_p = p - N_p$$

   a_p は楕円曲線の重要な値の一つである
4. $E \bmod p$ の群構造：
   - $E \bmod p$ 上の点集合は、有限アーベル群を形成する
   - この群は巡回群または 2 つの巡回群の直積として表される

また、楕円曲線上の点の位数の概念も学んだ。点 P の位数とは、$nP = O$（無限遠点）となる最小の正整数 n のことである。

この章の最後では、楕円曲線が暗号技術に応用されることに触れた。楕円曲線上の $E \bmod p$ の点 P,Q からとなる整数 n を求める問題 (ECDLP) の困難さが、楕円曲線暗号（ECC）の安全性の基礎となっていることを簡単に説明した。ECDLP を解くには、一般に $\mathcal{O}(\sqrt{N})$ ステップの処理が必要であり、P が 256 ビット以上ならば、ECDLP を実質的に解くことはできない。

この章の学習を通じて、楕円曲線の理論が有限体上でどのように展開されるかを理解した。特に、N_p や a_p の振る舞いが、楕円曲線の性質を反映していることを学んだ。これらの概念は、楕円曲線の類別や、より進んだ話題である楕円曲線の L 関数の研究にもつながっていくものである。

さらに、これらの理論が楕円曲線暗号（ECC）の基礎となっていることも理解した。ECC では、十分大きな素数 p を選び、$E \bmod p$ 上の点群の構造を利用して暗号システムを構築する。楕円曲線の離散対数問題が困難であることが、ECC の安全性の一つの根拠となっている。

この章の内容は、数論、代数幾何、暗号理論が美しく融合した例であり、純粋数学の理論が現代の情報セキュリティ技術に直接応用されている興味深い事例となっている。

（※文責: 伊澤叡遵）

4.2 ハードウェア実装の手法

4.2.1 開発環境

まずはじめに開発に必要なソフトウェアをダウンロードする。下記のリンクから、図 4.3 のインストーラをダウンロードする。バージョンは問わない。ダウンロードには AMD のアカウントが必要である。インストーラーを実行後、図 4.4 の画面で Vitis を選択。図 4.5 では使用する FPGA に準拠するものを選択する。その後はインストーラーの手順に沿ってダウンロードを実行する。
https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools.html

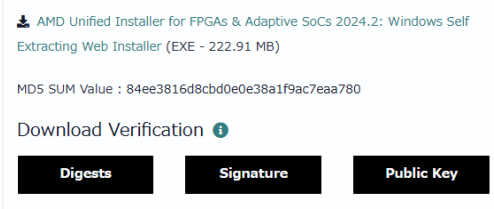


図 4.3　インストーラーの選択

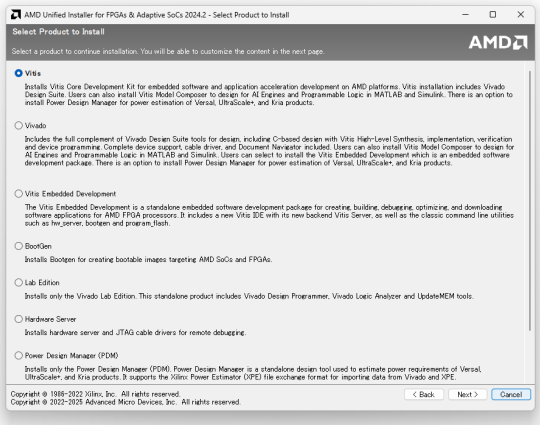


図 4.4　Vitis を選択

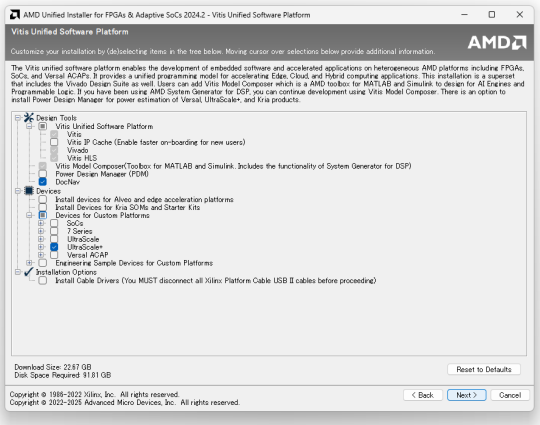


図 4.5　利用する FPGA の選択

インストールの完了後、Vitis の機能をすべて利用する為に Vitis HLS の開発者認証を行う。これにより、高位合成の結果を比較するために必要な Function Call Graph や Schedule Viewer の利用が出来る。http://www.xilinx.com/getlicense にアクセスしログインと個人情報を入力。図 4.6 の画面に遷移後、Vitis HLS License のチェックボックスを選択し、Generate Node-Locked License のボタンを押す。続いて Select a host からホストの追加を行う。図 4.7 の画面で OS の選択と Host ID の入力を行い ADD を選択。Windows の場合は図 4.8 のようにコマンドプロンプトで ipconfig /all で Howt ID Value に必要なイーサネットの MAC アドレスを確認することができる。ホストの追加後、Next を選択すると Xilinx, Inc から Xilinx.lic というファイルが添付されたメールが届くので、それを任意のフォルダに保存する。

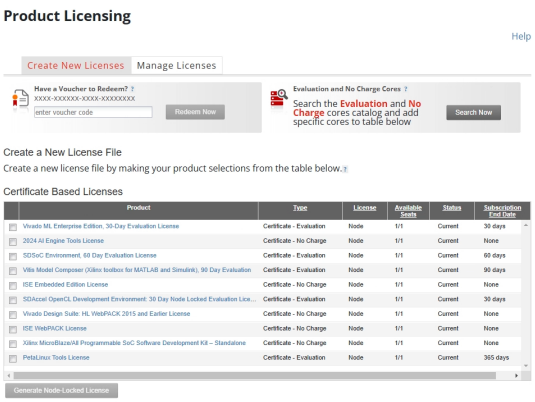


図 4.6　ライセンス認証画面

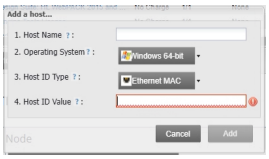


図 4.7　ホストの追加

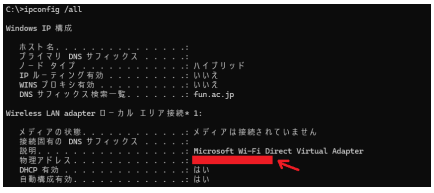


図 4.8　MAC アドレスの表示

ファイルを保存後、図 4.9 のアプリである Manage License を起動する。その後、左のタブの Load License を選択し、Copy License から先ほどダウンロードした Xilinx.lic を指定する。以上で開発環境のセットアップ完了である。

図 4.9　アプリの起動

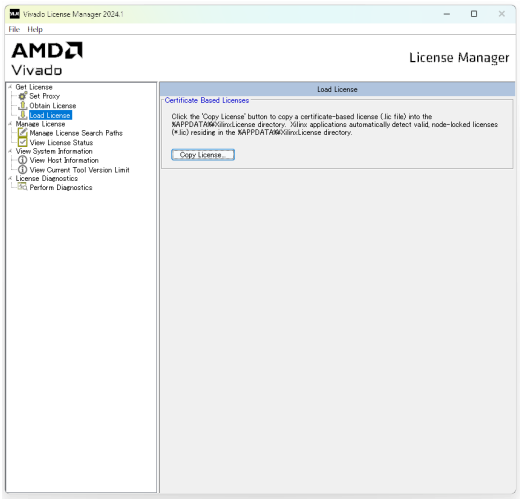


図 4.10　ライセンスの追加

（※文責: ディオガーディディラン基暉）

4.2.2　HLS コンポーネントの作成

図 4.11　Vitis アプリ

ハードウェア実装に向けて、まずは HLS コンポーネントを作成する。HLS コンポーネントでは、アルゴリズムの設計、C シミュレーション、高位合成、C/RTL 協調シミュレーションを行う。

HLS コンポーネントの作成にはまず、Vitis 統合 IDE を起動する (図 4.11)。起動後、ウェルカムページが表示されるので、HLS Development の欄から Create Component... を選択し、Create Empty HLS Componet を選択する (図 4.12)。ウェルカムページが表示されない場合はタイトルバーから File > New Component > HLS を選択する。続いて、任意の名前とフォルダを選択する (図 4.13)。

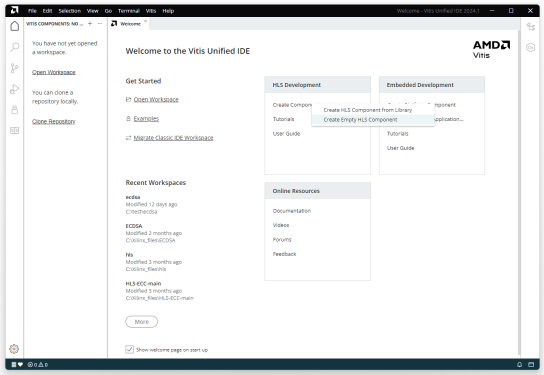


図 4.12　Create Empty HLS Component

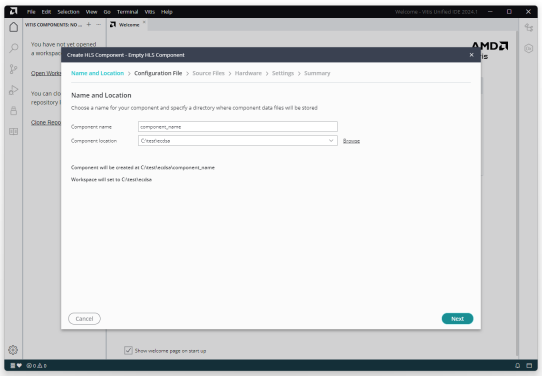


図 4.13　名前とファイルの指定

Configureation File と Source Files のページはデフォルトのまま進める (図 4.14)。Hardware のページでは使用する FPGA の型番を検索し、選択する (図 4.15)。Summary のページで構成を

確認し Finish を選択。以上で HLS コンポーネントの作成は終了である。

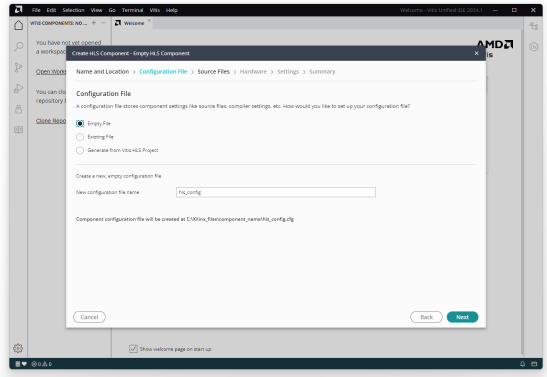


図 4.14　Configureation File

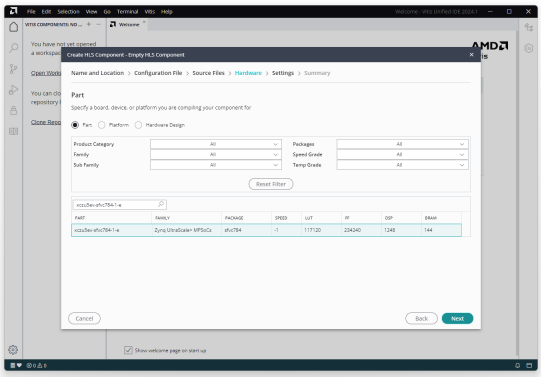


図 4.15　ハードウェアの指定

（※文責: ディオガーディディラン基暉）

4.2.3　アルゴリズムの設計

C 言語または C++ を用いてアルゴリズムの設計を行う。左のタブ上にある Source を右クリック、New Source File を選択し C/C++ ファイルを作成 (図 4.16)、ハードウェアに実装したいアルゴリズムをそのファイル内で定義する。高位合成を行うファイルには利用できる標準ライブラリの制限、HLS 特有の最適化用プラグマ、独自のデータを型などが存在する。例えば、楕円曲線暗号で利用する高 bit の数には、標準で 1024bit まで対応している任意精度型の ap_[u]int 型が “ap_int.h“ のインクルードで利用できる (図 4.17)。詳細は Xilinx の公式ドキュメント「Vitis HLS ライブラリ リファレンス」から確認することをおすすめする [5]。

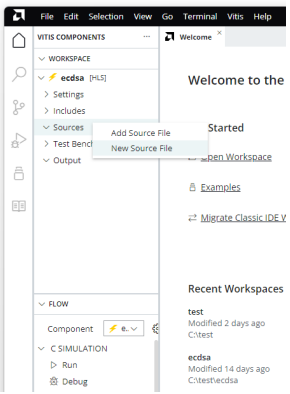


図 4.16　関数ファイルの追加

```
#include "ap_int.h"

void foo_top (…) {

 ap_int<9>  var1;          // 9-bit
 ap_uint<10>  var2;        // 10-bit unsigned
```

図 4.17　任意精度型を利用した例（出典：Xilinx)

（※文責: ディオガーディディラン基暉）

4.2.4　C シミュレーション

C シミュレーションを行うには、テストベンチファイルの作成が必要である。左のタブ上にある Test Bench を右クリック、New Source File を選択し C/C++ ファイルを作成 (図 4.18)、アルゴリズムをテストする為のコードをこのファイル内で定義する。テストベンチには main 関数を定義すること。テストベンチで使用されていない関数はシミュレーションされないことに注意する。続いて、Settings から hls_config.cfg を選択する。top という場所に移動し、高位合成を行う関数を指定する (図 4.19)。Browse ボタンをクリックすると関数が選択出来るので選択する (図 4.20)。

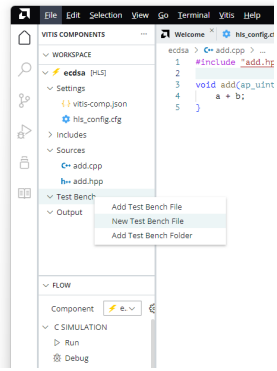


図 4.18　テストファイルの追加

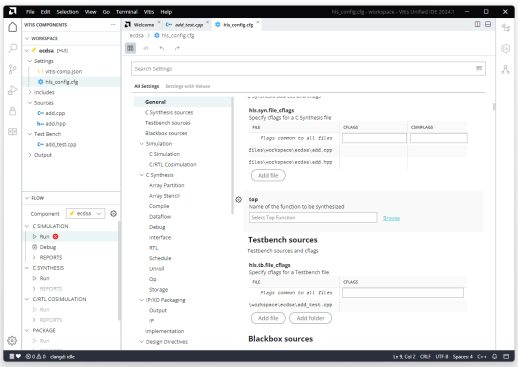


図 4.19　hls_config.cfg

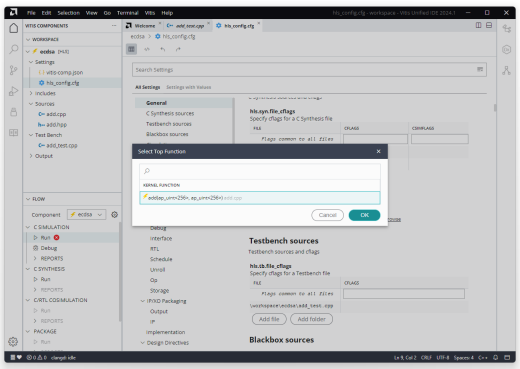


図 4.20　高位合成を行う関数の選択

続いて、実際に C シミュレーションを行う。左下のタブの FLOW の欄から、C Simulation の RUN ボタンをクリックする。すると、ターミナルが起動するので C-simulation finished successfully と表示されるまで待つ。無事に C シミュレーションが完了したらターミナル内の出力が、テストベンチで定義した出力の想定と一致するかを確認する。また、ここで高位合成を行う前に REPORTS の欄からコードアナライザーを用いてアルゴリズムを改善出来るかどうかを調査する (図 4.21)。ヒートマップを用いて演算負荷の高い部分を特定したり、結果を改善または最適化するための提案を確認することができる。

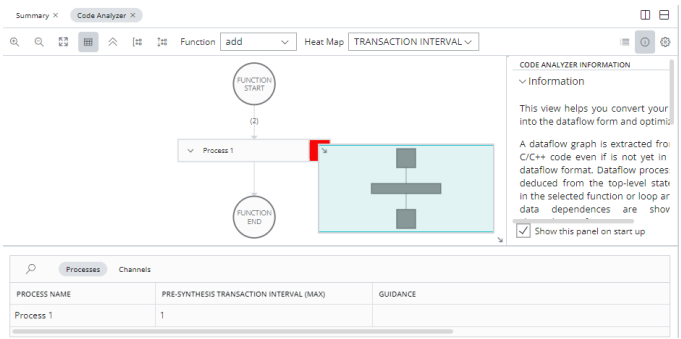


図 4.21　コードアナライザー

(※文責: ディオガーディディラン基暉)

4.2.5　高位合成

C シミュレーションが完了したら高位合成を行う。左下のタブの FLOW の欄から、C SYNTHESIS の RUN ボタンをクリックする。RUN の隣に緑のチェックマークが表示されたら高位合成の完了である。C シミュレーションと同様に REPORTS の欄からデザインが要件を満たしているかどうかを確認できる。また、左上の Output の欄からは作成した Verilog および VHDL の RTL ファイルが確認できる (図 4.22)。利用できるライブラリやプラグマの詳細については「Vitis 高位合成ユーザーガイド」を確認すると良い [5]。

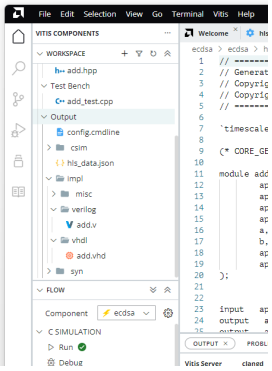


図 4.22　RTL ファイルの出力

（※文責: ディオガーディディラン基暉）

4.2.6　C/RTL 協調シミュレーション

C/RTL 協調シミュレーションを用いて、高位合成により作成したコードが C で設計したアルゴリズムと同じように動作するかを確認する。左下のタブの FLOW の欄から、C/RTL COSIMULATION ボタンをクリックする。C シミュレーションと同様にテストベンチのファイルをもとにテストを行う。また、シミュレーションが完了したら REPORTS の欄からデータフローについて確認できる。以上で Vitis 上での高位合成の流れは終了である。

（※文責: ディオガーディディラン基暉）

4.2.7　FPGA への書き込み

Vivado を利用してビットストリーム・ファイルを FPGA に書き込む。今回のプロジェクトではこのステップを完遂できなかった為、手順は概要だけである。Vivado 上では利用している開発ボードを選択し、Vitis で作成した Verilog ファイルを読み込む。その後、制約ファイル (XDC) を作成してビットストリーム・ファイルを生成する。開発ボードを利用している為、IP ブロックは自動生成することが可能であり、それを用いてファイルの作成が出来るはずである。FPGA を JTAG 接続し、書き込みを行う。

（※文責: ディオガーディディラン基暉）

第 5 章　結果

5.1　楕円曲線暗号の主要アルゴリズムの作成

前期では、暗号技術の基礎である数理の学習を行い、基盤となる知識を身に着けていった。後期は、それらを元に楕円曲線暗号のハードウェア実装へ向けて取り組んだ。目標である高位合成を用いた楕円曲線暗号のハードウェア実装による処理速度の確認を行うため、楕円曲線暗号の支配的な処理であるスカラー倍算のアルゴリズムの作成を行った。はじめにスカラー倍算のアルゴリズムを C 言語で作成し、動作確認を行った。続いて Vitis 上で、アルゴリズムを設計、アルゴリズムのシミュレーション、高位合成、ハードウェア記述言語で記述された回路のシミュレーションを行った。FPGA 本体での動作確認まではたどり着かなかったものの、楕円曲線暗号の主要な計算のシミュレーションは正確に行うことができた。今回作成したコードは付録として添付してある。

（※文責: ディオガーディディラン基暉）

5.2　アルゴリズムの高速化

スカラー倍算のアルゴリズムの動作確認の過程において、安全強度に比例する数 p を大きくするにつれて C シミュレーションを行うのが困難になる問題が生じた。問題解決の為にアルゴリズムの高速化を行った。Vitis 上のヒートマップを用いて処理時間を要している部分を確認をしたところ、スカラー倍算で利用される逆元計算のアルゴリズムに原因があった。シンプルなアルゴリズムを利用していた為、p が増加するにつれて指数関数的に処理時間を要するものであった。具体的には、安全強度 λ =128 の場合約 1.15×10^{77} 回の処理を要していた。これを解決する為に、逆元計算のアルゴリズムをバイナリ GCD 法を利用する物に変更した。これにより、処理時間は直線的な増加をする物に改善した。λ =128 の場合約 512 回の処理で済むようになった。 また、楕円曲線暗号において暗号化の支配的な処理は 2 回のスカラー倍算、復号の支配的な処理は 1 回のスカラー倍算であることから、結果的にアルゴリズムの高速化は楕円曲線暗号そのものを高速化したといえる。

（※文責: ディオガーディディラン基暉）

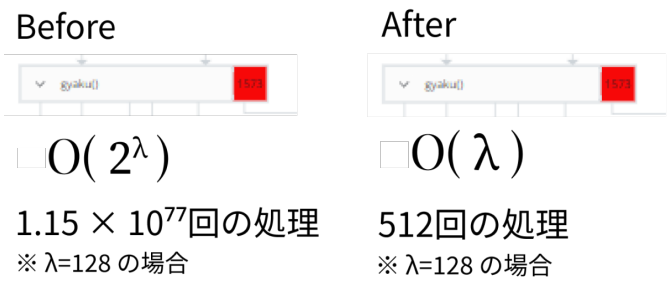


図 5.1　アルゴリズムの高速化

5.3　ポスターの作成

最終発表会に向けてポスターの作成を行った。

図 5.2　最終発表会ポスター

第 6 章　考察

6.1　スカラー倍算アルゴリズムの最適化による影響と意義

本研究では、楕円曲線暗号（ECC）のスカラー倍算アルゴリズムを mod p での逆元計算を行う手法に基づき最適化し、計算効率を大幅に向上させることができた。この結果は、ECC における主要な計算コストを削減する上で有効であったが、それにとどまらず、暗号化・復号処理全体の性能向上にも寄与することが示された。

スカラー倍算アルゴリズムの高速化がもたらした処理時間の直線的な増加特性は、安全強度が高まる場合でもシステムの実用性を維持する基盤となる。この点は、現代の情報セキュリティ環境において特に重要であり、次世代の高性能暗号システムの構築における基礎技術として位置付けられる。

6.1.1　安全強度に対する性能スケーラビリティの検討

アルゴリズムの改良により、安全性強度 $\lambda = 128$ の場合でも処理時間が現実的な範囲内に収まることが確認されたが、このスケーラビリティが他の安全強度設定や異なる楕円曲線においても同様に機能するかについては、さらなる検証が必要である。特に、安全強度の向上に伴う計算資源の消費バランスを考慮し、アルゴリズムのさらなる改良の可能性を検討する必要がある。

6.1.2　バイナリ GCD 法の適用限界

バイナリ GCD 法の適用により効率化が実現された一方で、楕円曲線暗号の異なるパラメータセットや計算環境において同様の成果を得られるかどうかは未解明の部分が多い。本手法が適用可能な領域を明確にし、場合によっては他の補完的なアルゴリズムの採用を検討することが、さらなる性能向上につながると考えられる。

6.2　高位合成を活用した設計手法の考察

本研究では、C 言語で作成したアルゴリズムを Vitis 上で高位合成することで効率的な開発が可能であった。この手法は、開発期間の短縮や設計の可視化において大きな利点を持つが、開発環境に依存した最適化の限界や、FPGA 実機での実行時の動作特性への影響を十分に評価する必要がある。特に、Vitis のヒートマップ機能を用いたボトルネックの特定が効率的であった一方、これが FPGA 上での物理的な動作特性とどの程度一致するかは、今後の詳細な評価が求められる。

6.3　FPGA 実機での動作確認の重要性

シミュレーションにおいてアルゴリズムの正確性を確認できたものの、FPGA 実機での動作確認が未完である点は課題として残る。FPGA 上での動作を検証することで、消費電力や処理速度、信頼性など、実運用を想定した詳細な評価が可能となる。これにより、理論上の改良が実用上どの程度効果を発揮するのかを明らかにすることが重要である。

6.4　総合的な考察と今後の課題

本研究の成果は、ECC のハードウェア実装における効率化の実現可能性を示したという点で意義がある。スカラー倍算の高速化を基盤として、点加算や点 2 倍算といった他の計算処理の最適化に着手することで、さらに実用的で高性能な暗号システムの開発が期待される。

一方で、実機動作確認や異なるアルゴリズムとの統合による全体最適化といった課題。これらに取り組むことで、研究成果の実運用への移行を加速させることが可能である。また、実機の実装まで行えなかった要因として、Vitis 周りのソフトウェアについて VitisIDE がここ数年で新しくなっており、ドキュメントがあまり充実しておらず苦労した点が挙げられる。

（※文責: 後藤瞭介）

参考文献

[1] はじめての数論 原著第 4 版 J.H. シルヴァーマン著 鈴木治郎訳

[2] Stoll, C., Klaaßen, L. Gallersdörfer, U. The carbon footprint of bitcoin. Joule 3, 1647–1661 (2019).

[3] FPGA 大全 第 2 版 小林優著

[4] Vitis 統合 IDE およびコマンドラインリファレンスマニュアル (UG1553), Advanced Micro Devices Inc., https://docs.amd.com/r/ja-JP/ug1553-vitis-ide, Jan. 2023 (accessed Jan. 2025).

[5] Vitis 高位合成ユーザーガイド (UG1399), Advanced Micro Devices Inc., https://docs.amd.com/r/2023.1-日本語/ug1399-vitis-hls, Jan. 2023 (accessed Jan. 2025).

付録 A　語録

AMD

AMD は、主に CPU や GPU を製造する半導体メーカーである。近年では、Xilinx の買収により FPGA 分野にも参入し、高性能コンピューティングや AI 分野での競争力を強化している。

LWE 方式、GGH 方式

LWE 方式と GGH 方式は、いずれも格子暗号の方式である。LWE 方式は学習誤差問題に基づき、GGH 方式は格子基底の短ベクトル問題を基盤としている。

Proof-of-work

Proof-of-work (PoW) とは、暗号通貨の取引を検証し、新しいブロックを生成するために計算作業を要求するアルゴリズムである。不正を防止し、システムの安全性を維持するために重要である。

RTL

RTL (Register Transfer Level) とは、デジタル回路の動作を記述する抽象化された手法である。論理ゲートレベルより高い抽象度を持ち、設計者はクロックサイクル単位でハードウェアの挙動を記述することができる。

Vitis

Vitis は、AMD(旧 Xilinx) が提供する統合開発環境であり、高位合成を通じて FPGA や SoC の効率的な開発をサポートする。C 言語ベースでの開発が可能で、設計の可視化やデバッグを容易にする。

Xilinx

Xilinx は、FPGA やプログラマブルデバイスの開発および製造を行う企業である。その製品は、多様な分野での応用が可能であり、2020 年に AMD によって買収され、事業領域を拡大した。

スカラー倍算

スカラー倍算とは、楕円曲線暗号において曲線上の点をスカラー値（整数倍）で演算する処理である。この計算は ECC の主要な演算であり、暗号化や署名において重要な役割を果たす。

プログラマブルロジック (PL)

プログラマブルロジック (PL) とは、電子回路の構成をプログラムによって動的に変更可能なデバイスである。具体例として FPGA が挙げられ、特定用途に応じた柔軟なハードウェア設計を実現する技術である。

バイナリ GCD 法

バイナリ GCD 法は、最大公約数（GCD）を効率的に計算するアルゴリズムである。この手法は加減算やシフト演算を用いるため、ハードウェアでの実装が容易で高速化に寄与する。

ビットコイン

ビットコインは、ブロックチェーン技術を基盤とした最初の分散型デジタル通貨である。中央管理者を必要とせず、個人間で安全かつ効率的な送金を実現する点が特徴である。

ブロックチェーン

ブロックチェーンとは、分散型台帳技術に基づき、取引情報を安全かつ不変な形で記録する仕組みである。この技術は中央集権的な管理を必要とせず、暗号通貨をはじめとする様々な分野で利用されている。

ヒートマップ

ヒートマップは、データの分布や負荷を視覚的に表現する手法である。Vitis では、設計中のボトルネック特定や最適化に活用される。

ハッシュ値

ハッシュ値とは、任意の入力データを固定長の出力に変換するハッシュ関数の結果である。この値は入力データに対して一意であり、データの改ざん検出や暗号技術において重要な役割を果たす。

マイニング

マイニングとは、暗号通貨のネットワークにおいて取引を検証し、ブロックを生成するプロセスである。この作業には計算資源が必要であり、成功報酬として新しい暗号通貨が与えられる仕組みである。

公開鍵暗号方式

公開鍵暗号方式とは、秘密鍵と公開鍵のペアを使用して暗号化および復号を行う方式である。これにより、通信の安全性が確保されるとともに、デジタル署名による認証機能も提供される。

格子暗号

格子暗号とは、高次元格子問題に基づく暗号方式である。量子コンピュータに対する耐性を有し、次世代の安全な暗号技術として注目されている。

高位合成

高位合成とは、高水準言語で記述したアルゴリズムをハードウェア記述言語（HDL）に変換する手法である。この技術により、開発期間の短縮と効率的な設計が可能となる。

楕円曲線暗号（ECC）

楕円曲線暗号（ECC）は、楕円曲線上の点の代数構造を利用した公開鍵暗号方式である。安全性の高さに対して鍵長が短いという特徴を持ち、特にリソースの限られた環境での使用に適している。

逆元計算

逆元計算とは、ある数値とその逆数を乗算した結果が 1 になるような値を求める計算である。特に、有限体における逆元計算は、暗号アルゴリズムの効率に大きく影響する。

有限体

有限体とは、有限個の要素を持つ代数体系であり、四則演算に閉じている。暗号技術では、計算対象を有限体に制限することで、効率的な演算を可能にしている。

付録 B　作成したコード

Listing B.1　ecdsa.h

```
#ifndef ECDSA_H
#define ECDSA_H

#include <ap_int.h>
#include <cstdint>
#include <utility>

#define BIT_NUM 1024
typedef ap_uint<BIT_NUM> ap_uint_t;

extern const ap_uint_t p;
extern const ap_uint_t A;

struct Point {
    ap_uint_t x;
    ap_uint_t y;
};

static ap_uint_t modp(const ap_uint_t& a);

ap_uint<9> countTrailingZeros(ap_uint<BIT_NUM> &x) ;
ap_uint_t gyaku(const ap_uint_t& a);
ap_uint_t gyaku_old(const ap_uint_t& a);

bool check_point(const Point& P1, const Point& P2);
Point point_add(const Point& P1, const Point& P2);
Point point_dbl(const Point& P1);
Point point_mult(const Point& P, const ap_uint_t& n);

void ecdsa(Point P1, Point P2, bool is_add, Point &result);

#endif // ECDSA_H
```

Listing B.2　ecdsa.cpp

```
#include "ecdsa.h"
#include <cctype>
#include <iostream>

const ap_uint_t p = ap_uint_t("
    ffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f",16);
const ap_uint_t A = 15;
```

```
static ap_uint_t modp(const ap_uint_t &a) { return a % p; }

ap_uint_t gyaku(const ap_uint_t &a) {
#pragma HLS INLINE
  ap_uint_t u = a;
  ap_uint_t v = p;
  ap_uint_t x1 = 1;
  ap_uint_t x2 = 0;

BINARY_GCD_LOOP:
  while (u != 1 && v != 1) {
#pragma HLS PIPELINE II = 1

    while ((u & 1) == 0) {
      u >>= 1;
      if (x1 & 1)
        x1 += p;
      x1 >>= 1;
    }

    while ((v & 1) == 0) {
      v >>= 1;
      if (x2 & 1)
        x2 += p;
      x2 >>= 1;
    }

    if (u >= v) {
      u -= v;
      if (x1 >= x2)
        x1 -= x2;
      else
        x1 = p - (x2 - x1);
    } else {
      v -= u;
      if (x2 >= x1)
        x2 -= x1;
      else
        x2 = p - (x1 - x2);
    }
  }

  return (u == 1) ? x1 : x2;
}

ap_uint_t gyaku_old(const ap_uint_t &a) {
  ap_uint_t b = 1;
```

```
  for (ap_uint_t i = 1; i < p - 1; i++) {
    b = b * a;
    b = modp(b);
  }
  return b;
  // #pragma HLS PIPELINE II=1
  // ap_uint_t x = 1;
  // ap_uint_t result = 0;

  // for (x = 1; x < p; x++) {
  // ap_uint_t prod = modp(a * x);
  // result = (prod == 1) ? x : result;
  // }
  // return result;
}

bool check_point(const Point &P1, const Point &P2) {
  if (P1.x == P2.x && P1.y == P2.y) {
    return true;
  }
  return false;
}

Point point_add(const Point &P1, const Point &P2) {
  ap_uint_t a, b, x3, y3;
  ap_uint_t x1 = P1.x;
  ap_uint_t x2 = P2.x;
  ap_uint_t y1 = P1.y;
  ap_uint_t y2 = P2.y;

  if (check_point(P1, P2)) {
    return Point{0, 0};
  }

  a = (x1 - x2 + p);
  a = modp(a);
  a = gyaku(a);
  b = ((y1 - y2 + p) * a);
  b = modp(b);

  x3 = (b * b - x1 - x2 + 2 * p);
  x3 = modp(x3);

  y3 = (b * (x1 - x3 + p)) - y1 + p;
  y3 = modp(y3);

  Point P3 = {x3, y3};
  return P3;
```

```
}

Point point_dbl(const Point &P1) {
  ap_uint_t a, b, x3, y3;
  ap_uint_t x1 = P1.x;
  ap_uint_t y1 = P1.y;

  a = (y1 + y1);
  a = modp(a);
  a = gyaku(a);
  b = ((3 * x1 * x1 + A) * a);
  b = modp(b);

  x3 = (b * b - x1 - x1 + 2 * p);
  x3 = modp(x3);

  y3 = (b * (x1 - x3 + p)) - y1 + p;
  y3 = modp(y3);

  Point P3 = {x3, y3};
  return P3;
}

Point point_mult(const Point &P, const ap_uint_t &n) {
  Point R = P;
  int t = BIT_NUM - 1;
  if (n == 0) {
    printf("n was 0");
    return Point{0, 0};
  }
  while (n[t] == 0) {
    t--;
  }
  for (int i = t - 1; i >= 0; i--) {
    std::cout << n[i] << std::endl;
    R = point_dbl(R);
    if (n[i] == 1) {
      std::cout << "point add" << std::endl;
      R = point_add(R, P);
    }
  }
  return R;
}

void ecdsa(Point P1, Point P2, bool is_add, Point &result) {
#pragma HLS INTERFACE s_axilite port = P1
#pragma HLS INTERFACE s_axilite port = P2
#pragma HLS INTERFACE s_axilite port = is_add
```

```
#pragma HLS INTERFACE s_axilite port = result
#pragma HLS INTERFACE s_axilite port = return

  if (is_add) {
    // Point addition
    result = point_add(P1, P2);
  } else {
    // Point doubling (when P1 = P2)
    result = point_dbl(P1);
  }
}
```

Listing B.3 test.cpp

```
#include "ecdsa.h"
#include <iostream>
#include <stdio.h>



int main(void){
    ap_uint<5> n ("18", 10) ;
    printf("n[0]:");
    std::cout << n[0]<< std::endl;
    printf("n[0]:");
    std::cout << n[0]<< std::endl;
    printf("n[1]:");
    std::cout << n[1]<< std::endl;
    printf("n[2]:");
    std::cout << n[2]<< std::endl;
    printf("n[3]:");
    std::cout << n[3]<< std::endl;
    printf("n[4]:");
    std::cout << n[4]<< std::endl;
    std::cout << n << std::endl;

    if(n[0] == true){
        printf("abc");
    }

        ap_uint_t X[18];
        ap_uint_t Y[18];
        ap_uint_t x1,x2,x3,x4,x4_1,x5;
        ap_uint_t y1,y2,y3,y4,y5;
        ap_uint_t r1,r2,r3;
        ap_uint_t A=15;
        ap_uint_t B=8;

        //P4=(P1+P2)+P3;
```

```
    //P5=P1+)P2+P3);

    srand(18);

    X[0]=0;
    X[1]=0;
    X[2]=4;
    X[3]=4;
    X[4]=5;
    X[5]=5;
    X[6]=6;
    X[7]=6;
    X[8]=10;
    X[9]=10;
    X[10]=11;
    X[11]=11;
    X[12]=14;
    X[13]=14;
    X[14]=15;
    X[15]=15;
    X[16]=16;
    X[17]=16;
    Y[0]=5;
    Y[1]=12;
    Y[2]=8;
    Y[3]=9;
    Y[4]=2;
    Y[5]=15;
    Y[6]=5;
    Y[7]=12;
    Y[8]=16;
    Y[9]=11;
    Y[10]=5;
    Y[11]=12;
    Y[12]=2;
    Y[13]=15;
    Y[14]=2;
    Y[15]=15;
    Y[16]=3;
    Y[17]=14;

    // r1=rand()%18;
    // r2=rand()%18;
    // r3=rand()%18;
    r1=5;
r2=11;
r3=15;
std::cout << "r1=" << r1 << ", r2=" << r2 << ", r3=" << r3 << std::endl
```

```
    ;

    x1=X[r1];
    x2=X[r2];
    x3=X[r3];
    y1=Y[r1];
    y2=Y[r2];
    y3=Y[r3];

std::cout << "P1=(" << x1 << ", " << y1 << ")" << std::endl;
std::cout << "P2=(" << x2 << ", " << y2 << ")" << std::endl;
std::cout << "P3=(" << x3 << ", " << y3 << ")" << std::endl;


    // x4=point_dblx(x1,y1,A);
// y4=point_dbly(x1,y1,A);
Point P1 = {x1, y1};
Point P2 = {x2, y2};
Point P3 = {x3, y3};
Point P4, P4_1, P5, P5_1;

P4 = point_add(P1, P2);
P4_1 = point_add(P3, P4);
P5 = point_add(P2, P3);
P5_1 = point_add(P1, P5);

std::cout << "P4=(" << P4.x << ", " << P4.y << ")" << std::endl;
std::cout << "P4_1=(" << P4_1.x << ", " << P4_1.y << ")" << std::endl;
std::cout << "P5=(" << P5.x << ", " << P5.y << ")" << std::endl;
std::cout << "P5_1=(" << P5_1.x << ", " << P5_1.y << ")" << std::endl;
    // x4_1=point_addx(x3,y3,x4,y4);
    // y4=point_addy(x3,y3,x4,y4);
    // printf("P4=(%d, %d)\n",x4_1,y4);

// unsigned int x5_1;
    // x5=point_addx(x2,y2,x3,y3);
    // y5=point_addy(x2,y2,x3,y3);
    // printf("P5'=(%d, %d)\n",x5,y5);
    // x5_1=point_addx(x1,y1,x5,y5);
    // y5=point_addy(x1,y1,x5,y5);
    // printf("P5=(%d, %d)\n",x5_1,y5);

Point P_6, P_9, P_6_9, P_9_6;
P_6 = point_mult(P1, 6);
P_9 = point_mult(P1, 9);

ap_uint_t g, g0;
g = gyaku(15);
```

```
    //g0 = gyaku_old(15);
    std::cout << "gyaku=(" << g << ")" << std::endl;
    //std::cout << "gyaku_old=(" << g0 << ")" << std::endl;

    P_6_9 = point_mult(P_6, 9);
    P_9_6 = point_mult(P_9, 6);
    std::cout << "P_6_8=(" << P_6_9.x << ", " << P_6_9.y << ")" << std::
        endl;
    std::cout << "P_9_6=(" << P_9_6.x << ", " << P_9_6.y << ")" << std::
        endl;

    std::cout << "P_6=(" << P_6.x << ", " << P_6.y << ")" << std::endl;
    std::cout << "P_9=(" << P_9.x << ", " << P_9.y << ")" << std::endl;
        return 0;
}
```