# 最先端暗号のハードウェア実装

藤本 大介[1]　坂本 純一[1]　奥秋 陽太[1]　吉田 直樹[1]　松本 勉[1,a)]

**概要**：サイバーフィジカルシステムやクラウド活用の進展に伴い、公開鍵証明書を用いずに ID を公開鍵として暗号通信が行える、多数のディジタル署名を集約して伝送し一括検証が行える、暗号化したままデータ検索が行えるなど、従来の公開鍵暗号技術より機能性を高めた高機能暗号技術への期待が高まっている。高い機能性の代償として計算は複雑となる傾向があるため、高機能暗号の高速実装、低消費エネルギー実装にはハードウェアが有用である。本発表では、高機能暗号を含む最先端暗号のハードウェア実装技術について述べる。

# Hardware Implementation of the State-of-the-Art Cryptgraphy

Daisuke Fujimoto[1]　Junichi Sakamoto[1]　Yota Okuaki[1]　Naoki Yoshida[1]
Tsutomu Matsumoto[1,a)]

***Abstract:*** One of the biggest problems of the emerging cyber-physical and cloud computing systems is how to ensure security with energy efficiency. As a solution to the problem there is a growing expectation of adopting advanced cryptography with rich functionalities such as (1) ID-based encryption which does not require public key of the intended recipient of the ciphertext, (2) Aggregate signature by which aggregation of multiple digital signatures and their collective verification can be possible, (3) Searchable encryption which enables direct data retrieval over encrypted database without decrypting the database, and so on. Thus dedicated hardware is useful for high-speed mounting of the advanced cryptographic schemes and low energy consumption because computation tends to be complicated as compensation for rich functionalities. In this paper, we describe the hardware implementation technology for advanced cryptography which is based on pairings defined over elliptic curves.

## 1. はじめに

近年、サイバーフィジカルシステムやクラウド技術の進展に伴い、ネットワークを介してやり取りされるデータが増大している。それらのデータには他人には非公開にすべき個人情報などの秘匿情報が含まれる。また、産業用システムや社会インフラにおける制御コマンドなどが改ざんされることを防ぐ必要がある。データの守秘や改ざん防止には暗号技術が有用であるが、データ量の増大に伴って暗号処理にかかる計算時間が長くなり、処理性能の劣化や消費電力の増大を引き起こしている。これらの課題を解決するためには、既存の暗号化を行うための専用ハードウェアの高速化や省電力化だけでなく、暗号方式自体の改善を図ることが求められている。そこで、公開鍵証明書を用いずに ID を公開鍵として暗号通信が行える、多数のディジタル署名を集約して伝送し一括検証が行える、暗号化したままデータ検索が行えるなど、従来の公開鍵暗号技術より機能性を高めた高機能暗号技術への期待が高まっている。一方で、高い機能性の代償として計算は複雑となる傾向があるため、高機能暗号の高速実装、低消費エネルギー実装にはハードウェアが有用である。

本稿では、高機能暗号を実現するためのペアリング計算に着目し、その複雑な計算を高速化するためのハードウェア実装について述べる。

以下に、本稿の構成を示す。２章では、ペアリング暗号実装に必要な計算について述べる。３章では現在提案されているペアリング計算の実装手法についてまとめ、４章で

[1] 横浜国立大学
Yokohama National University, Yokohama, Kanagawa 240–8501, Japan
[a)] tsutomu@ynu.ac.jp

ハードウェア実装手法のうちパイプライン型剰余乗算器を用いた高速実装について述べ、5章でFPGAによる実装評価結果を示し、6章にてまとめる。

## 2. ペアリング暗号実装の事前準備

本節では、ペアリング計算の中で現在最も効率が良いとされているOptimal Ateペアリングについて定義及び計算手順を示す。

BN曲線 $E : Y^2 = X^3 + b$ は埋め込み次数 k=12 の素体 $\mathbb{F}_p$ 上で定義される楕円曲線である。本稿では $\mathbb{F}_p$-有理点群の位数を r,p 乗のフロベニウス写像を $\pi_p$ と表記する。BN曲線Eが定義されている時、E上の二つの部分群 $\mathbb{G}_1 = E[r] \cap \ker(\pi_p - [1]) = E(\mathbb{F}_p)[r]$,$\mathbb{G}_2 = E[r] \cap \ker(\pi_p - [p]) \subseteq E(\mathbb{F}_p^{12})[r]$ とし、有限体 $\mathbb{F}_p^{12}$ 上の1のr乗根を $\mathbb{G}_3 = \mu_r \in \mathbb{F}_{p^{12}}^*$ とするときOptimal Ateペアリングを次のように定義する。

$$a_{\mathrm{opt}} : \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_3 \qquad (1)$$
$$(Q, P) \mapsto (f_{6u+2,Q}(P) \cdot g_Q(P))^{\frac{p^{12}-1}{r}}.$$

ここで、$f_{s,Q}$ は $\mathrm{div}(f_{s,Q}) = s(Q) - ([s]Q) - (s-1)(\mathcal{O})$ となるE上の有理関数でMiller関数と呼ばれる。P、Qはそれぞれ $\mathbb{G}_1, \mathbb{G}_2$ 上の点である。$g_Q$ はE上の点Q,Q'を通る直線 $l_{Q,Q'}$ を用いて次のように定義されるE上の多項式関数である。

$$g_Q = l_{[6u+2]Q,\pi_p(Q)} \cdot l_{[6u+2]Q+\pi_p(Q),-\pi_p^2(Q)} \qquad (2)$$

k=12 の時のBN曲線は小椋らが証明した正規化が不要な楕円曲線の条件に含まれている [1]。

アルゴリズム1にOptimal Ateペアリングの計算方法を示す。現在、効率よく実装可能といわれているものは、254ビットのBN曲線を使用した実装であり。その際のパラメータは $u = -(2^{62} + 2^{55} + 1)$ である。ここで、アルゴリズム中の計算量を考えるために名称を付ける。1〜7行目のMillerのアルゴリズム部分をMillerLoopとする。8〜11行目の $g_Q$ を求める部分をFinal additionとする。12行目の最終べきは計算量の評価のため、$(p^6 - 1)$ 乗、$(p^2 + 1)$ 乗と分割し、残りの $(p^4 - p^2 + 1)/r$ 乗をHard partとする。

ペアリング計算はこれらの各計算をいかに効率よく実装するかがポイントであり、乗算器などの回路の高性能化だけでなく、これらの計算の計算アルゴリズムの最適化も重要となる。

## 3. 関連研究

本節では、ペアリング暗号実装についてこれまでに提案されているものについてまとめる。ハードウェア実装だけでなく、CPU上で計算されるソフトウェア実装についても取り上げる。ソフトウェア実装においては、Intel Core

**Algorithm 1:** BN曲線上のOptimal Ateペアリング [7]

**input :** $P \in \mathbb{G}_1, Q \in \mathbb{G}_2, n = |6u+2|$, where $u < 0$.
**output:** $a_{opt}(Q, P)$.

1 $T \leftarrow Q, f \leftarrow 1$;
2 **for** $i = \lfloor \log_2 n \rfloor - 1$ ***downto*** $0$ **do**
3 $\quad f \leftarrow f^2 \cdot l_{T,T}(P), T \leftarrow 2T$;
4 $\quad$ **if** $n_i = 1$ **then**
5 $\quad\quad f \leftarrow f \cdot l_{T,Q}(P), T \leftarrow T + Q$
6 $\quad$ **end**
7 **end**
8 $Q_1 \leftarrow \pi_p(Q), Q_2 \leftarrow \pi_p^2(Q)$;
9 $T \leftarrow -T, f \leftarrow f^{p^6}$;
10 $f \leftarrow f \cdot l_{T,Q_1}, T \leftarrow T + Q_1$;
11 $f \leftarrow f \cdot l_{T,-Q_2}, T \leftarrow T - Q_2$;
12 $f \leftarrow f^{(p^6-1)(p^2+1)(p^4-p^2+1)/r}$;
13 **return** $f$

等の汎用CPUを用いて高速なペアリング計算を実現している [2][3][4]。CPUに搭載されている演算回路 (ALU) のデータ幅は64ビットが標準的であるので、このデータ幅を基準に高基数のデータの加算や乗算を、ALUを複数回使用することで計算する。具体的には筆算方式やKaratsuba法を用いて構成する。文献 [3] の成果により、Karatsuba法では乗算回数は減るものの加算回数が増え、CPUに搭載されているALUにおいては加算と乗算にかかるクロックサイクル数の差がないことから、トータルの演算回数が少ない筆算方式のほうが高速であると報告されている。また、ソフトウェア実装の効率はCPU自体に用意されている命令コードに大きく左右され、CPUの命令セット及びアーキテクチャの世代ごとに性能が異なる。さらに、汎用動作のためのフレキシブルなメモリ操作などの暗号処理以外の動作が含まれるため、消費電力の観点からは不利となる。また、汎用CPUを占有し続け暗号計算のみに用いることは現実的でないため、他の処理の割り込みが入るほどに暗号計算の性能は劣化する。

ハードウェア実装においてはソフトウェア実装と異なり、ペアリング計算に適した演算器を専用に設けることができる。特に多用される剰余乗算が時間のかかる計算であるため、剰余乗算を行うための回路の改良が主眼となっている。剰余乗算器の構成方法には大きく分けてペアリング計算のデータサイズそのままの高基数の乗算器を設ける手法 [5][8][9][11][12] と、RNS(Residue Number System) という数系を利用して乗算器のサイズを小さくする手法 [6][7] が存在する。RNSを用いた実装は高速かつ低エネルギーでペアリング演算を実現することができるが、基底変換処理を追加で行う必要があり、必要なクロックサイクル数を減らすことは困難である。一方で、高基数乗算器を用いる実装では、剰余乗算を乗算と剰余を別に設ける [5]、剰余乗

算全体でパイプラインを構成する [11]、拡大体上での乗算全体で最適化を行う [8][12] など様々な最適化が可能であり研究が盛んに行われている。

それぞれの実装の具体的な性能の比較については 5 章で行い、これらのハードウェア実装のうち剰余乗算のパイプライン化を行いスループットを向上させ、ペアリング計算の高速化を図った実装について次章以降で述べる。

## 4. パイプライン型剰余乗算器アーキテクチャ

本節では、前節で取り上げたハードウェア実装のうち我々の研究成果であるパイプライン型剰余乗算器を用いたアーキテクチャでのペアリング計算について示す。この実装は 1 回のペアリング計算の高速化を狙った実装である。

### 4.1 アーキテクチャ

ペアリング計算のアルゴリズムを実装するための $\mathbb{F}_p$ 上の演算の構成自体はソフトウェア実装 [3] とほぼ同じである。ソフトウェア実装とハードウェア実装の違いは、ソフトウェア実装では $\mathbb{F}_p$ 上の剰余乗算を実装する際に乗算と剰余を分けて実装するのに対して、ハードウェア実装では剰余乗算の専用回路を設ける点である。ハードウェア実装に適した拡大体での演算の構成は Yao らの手法を用いた [7]。

図 1 にペアリング計算を行うための回路全体のブロック図を示す。$\mathbb{F}_p$ 上の乗算にはパイプライン型モンゴメリ乗算器 [11] を用いる。パイプライン型モンゴメリ乗算器は 1 クロックサイクルごとに別のデータを入れることのできる高速な乗算器である。逆元計算はペアリング中に一回しか存在しないが、既存研究 [7] では 20 %程度の処理時間を占めるため、専用回路を導入し高速化を図る。

ペアリング計算を行う上で加算は拡大体の多項式計算部分に多く用いられるので、剰余乗算器の前後に Pre-adder や Post-adder を接続することで、前段のデータ保持と加算を効率的に実現できる [7]。また、楕円曲線上の加算などで頻出する 6 以下の定数で乗じる回路 (Constant multiplier) を剰余乗算器の後段に挿入することで、乗算の回数を減らすことができる。

データの保持は 2 個の 318 bit メモリを用いる。254 bit でない理由はモンゴメリ乗算に Quotient Pipelining を用いているためであり、詳細は次節で示す。乗数と被乗数 2 個を同時に読み出すには演算器の 2 倍の速度のメモリが必要となるため設計の容易さから、2 個のメモリを設置し、メモリへの入力を選択できる回路を設けた。これらの演算器をシーケンサを用いて制御し、CPU など外部インターフェースからの入力データなどはシーケンサを通してメモリへと格納され、演算結果はシーケンサを通じて外部へと出力される。

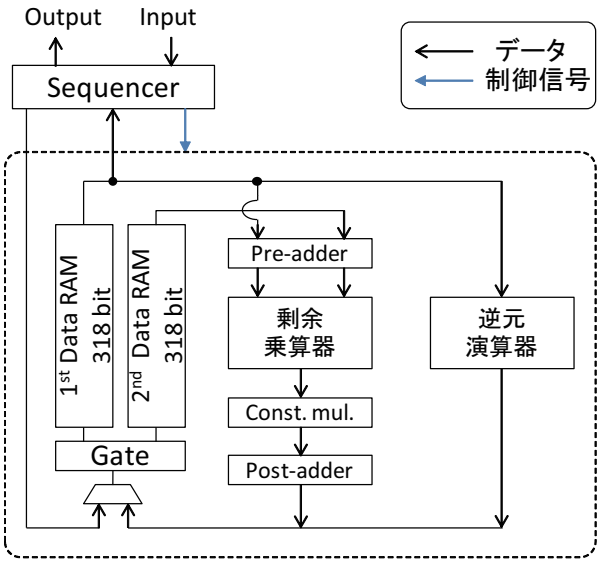


図 1 アーキテクチャブロック図

#### 4.1.1 パイプライン型剰余乗算器

高速に剰余乗算を行うためのアルゴリズムとして低レイテンシ実装である Quotient Pipelining 方式のモンゴメリ乗算を選択した。アルゴリズムをアルゴリズム 2 に示す。パイプライン型モンゴメリ乗算器は Quotient Pipelining のアルゴリズムのループ部分を展開し、ループの各段を全て別々の回路として実装したものである [11]。ループ部を展開しないで同じ回路を再利用した実装（ループ型実装とする）の場合、計算が終わりデータが出力されるまで待たないとならないが、展開した場合は待つ必要はなく連続して計算したいデータを入力することができるためスループットが高まる。ループ型の剰余乗算器のデータフローを図 2 に、パイプライン型剰余乗算器のデータフローを図 3 に示す。パイプライン型と同じ性能を満たすためには、ループ型回路を複数個設けて入力を切り替えて運用することが考えられる。しかし、318 bit のデータを並列数だけ処理する選択回路を実装するコストは大きくなる。さらに図 3 に示すようにアルゴリズム上で初期値が 0 の項には回路を設けないでよいため展開後最適化を行うと回路規模が小さくなることがわかる。これによりパイプライン型にすることで回路オーバーヘッドが少なく高性能な剰余乗算器を実現できるといえる。

ペアリングに適したパイプライン型剰余乗算器を設計するためにはパラメータの選択が重要である。パイプラインに入れられるデータは互いに依存しないものに限られ、パイプラインの段数を合わせる必要がある。むやみにパイプライン段数を大きくしても、パイプライン空きが発生してしまい性能が下がる。今回の実装では Yao らの実装 [7] と同様に剰余乗算器を通るパスのパイプライン段数を 18 とした。内訳としてはメモリ内 2 段、pre-adder1 段、剰余乗算器のループ部 12 段、剰余乗算器の出力段で 1 段、

**Algorithm 2:** Quotient pipelining モンゴメリ乗算 [10]

**input :** 分割幅:$k$; 遅延定数:$d$; ブロック数:$n$; 被乗数:$A$; 乗数:$B$; 法:$M$; $M > 2$, $\gcd(M, 2) = 1$, $(-MM') \bmod 2^k = 1$, $\tilde{M} = (M' \bmod 2^{k(d+1)})M$, $4\tilde{M} < 2^{kn} = R$, $M'' = (\tilde{M} + 1)/2^{k(d+1)}$, $0 \le A, B \le 2M$, $B = \sum_{i=0}^{n+d} (2^k)^i b_i$, $b_i \in \{0, 1, \cdots, 2^k - 1\}$, and $b_i = 0$ for $i \ge n$

**output:** $S_{n+d+2} \equiv ABR^{-1} (\bmod M), 0 \le S_{n+d+2} \le 2\tilde{M}$.

$S_0 := 0$; $q_{-d} := 0; \cdots; q_{-1} := 0$ ;

**for** $i := 0$ ***to*** $n + d$ **do**

  $L_1 : q_i \quad := S_i \bmod 2^k$ ;

  $L_2 : S_{i+1} := S_i/2^k + q_{i-d}M'' + b_i A$ ;

**end**

$S_{n+d+2} := 2^{kd} S_{n+d+1} + \sum_{j=0}^{d-1} q_{n+j+1} 2^{kj}$;

**return** $S_{n+d+2}$

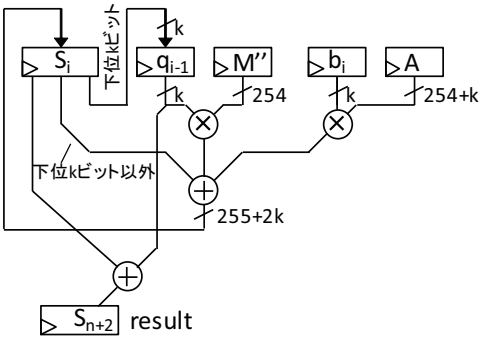


**図 2** ループ型実装のブロック図

Constant multiplier を合わせて 1 段、post-adder1 段である。この時のパイプライン型剰余乗算器のパラメータは分割 bit 幅 $k = 32$、遅延定数 $d = 1$、ブロック数 $n = 10$ である。この時 $R = 2^{kn}$ より $R$ は 320 bit となり、データサイズは高々 318 bit となる。このパラメータは 254 bit の BN 曲線上の Optimal Ate ペアリングに最適化したものであり、他の曲線ではパラメータやパイプライン段数を再考する必要がある。

### 4.1.2 $\mathbb{F}_p$ 逆元演算器

有限体上の逆元を求めるアルゴリズムとして、フェルマーの小定理か拡張ユークリッド互助法を用いるのが一般的である。両者とも計算量のオーダーは等しいが、フェルマーの小定理は剰余乗算を基本演算とするため、計算にかかるサイクル数が多い。そこで、拡張ユークリッド互助法を用いた逆元演算用の専用回路を設けることを検討した。

拡張ユークリッド互助法は、アルゴリズム 3 に示されるように、2 による除算で構成するとシフトを用いて簡単に計算でき、回路も小さく実装することができる。このアルゴリズムでの計算はモンゴメリ形式ではなく通常の数値形式に戻して行う必要がある。そのため、モンゴメリリダクション (剰余乗算 1 回) を行って通常の数値形式に戻したあと、逆元演算、モンゴメリ形式への変換 (剰余乗算 1 回) の順で計算を行う。

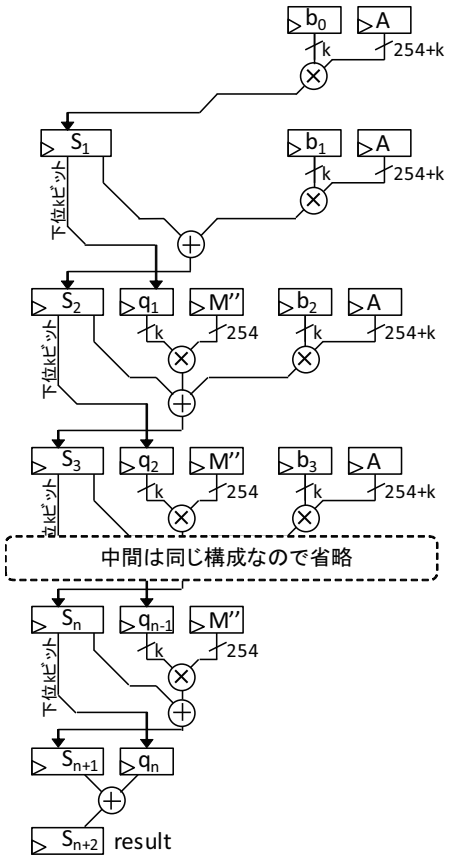


**図 3** アンロールドパイプライン型実装のブロック図

**Algorithm 3:** バイナリ拡張ユークリッド互助法

**input :** 法 $p$ and $a \in [1, p - 1]$.

**output:** $a^{-1} \bmod p$.

$u \leftarrow a, v \leftarrow p$;

$x_1 \leftarrow 1, x_2 \leftarrow 0$;

**while** $u \ne 1$ *and* $v \ne 1$ **do**

  **while** $u$ *is even* **do**

    $u \leftarrow u/2$;

    **If** $x_1$ is even **then** $x_1 \leftarrow x_1/2$; **else** $x_1 \leftarrow (x_1 + p)/2$;

  **end**

  **while** $v$ *is even* **do**

    $v \leftarrow v/2$;

    **If** $x_2$ is even **then** $x_2 \leftarrow x_2/2$; **else** $x_2 \leftarrow (x_2 + p)/2$;

  **end**

  **if** $u \ge v$ **then** $u \leftarrow u - v$;

  **else** $x_1 \leftarrow x_1 - x_2$;

**end**

**if** $u = 1$ **then return** $x_1$ *mod* $p$;

**else return** $x_2$ *mod* $p$;

### 4.1.3 多項式計算用回路群

多項式計算を効率的に行うために、剰余乗算を行う前に

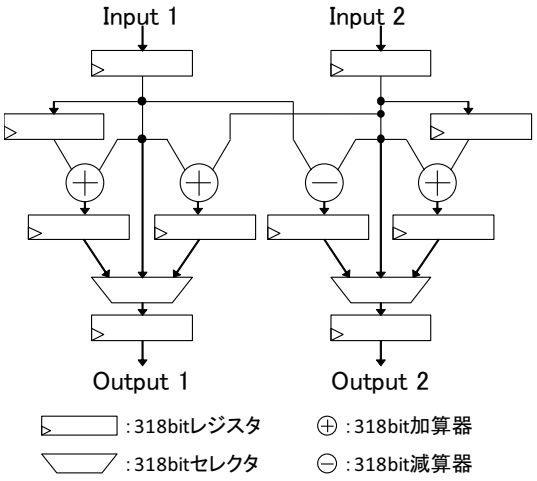


図 4　Pre-adder ブロック図

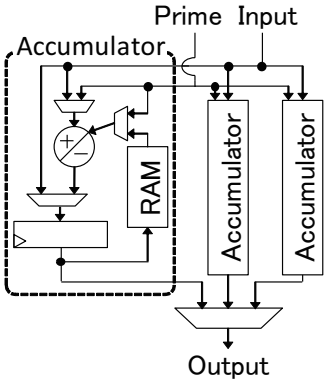


図 5　Post-adder ブロック図

項同士を加減算する Pre-adder（図 4）と剰余乗算後に以前に計算した項との加減算を行うための Post-adder（図 5）を設けている。Post-adder は 8 個のデータを格納する程度の小さい RAM を設け任意のデータとの加減算をすることで、乗算器へのデータ通過回数を減らし、全体の計算サイクル数を削減している。

## 5. FPGA による実装評価

### 5.1 評価環境・評価指標

ハードウェアのプロトタイプ環境として、本稿では Xilinx 社 KCU105 ボードを選択した。KCU105 ボードには算術演算に用いる DSP(Digital Signal Processor) が 1920 個、8 bit の桁上げ付き演算器が 30300 個、論理回路に用いる 6 bitLUT(Lookup Table) が 424200 個、データを保持するための FF(Flip Flop) が 484800 個搭載されている。 DSP には 27 bit × 18 bit の符号付乗算器が含まれている。　開発ソフトウェアは Vivado 2017.1 である。

開発ソフトにより得られる設計結果としては回路規模、最大回路遅延がある。回路規模は使用した DSP、LUT、FF などの素子の数であり、最大回路遅延は、FF 間の最大遅延であり、ロジック部分の遅延と配線による遅延で構成される。演算器としての性能は回路遅延により決定される。最大回路遅延よりも速い動作クロックでは正しい結果が返ってこないため、動作周波数は最大回路遅延を考慮し十分低くする必要がある。

表 1　ペアリングハードウェア使用回路内訳

| 部分名 | DSPs | LUTs | FFs | CARRY8s | RAM |
|---|---|---|---|---|---|
| シーケンサ | 0 | 7271 | 1212 | 516 | 5 |
| Pre-adder | 0 | 3687 | 2323 | 336 | 0 |
| 剰余乗算器 | 460 | 21782 | 12686 | 2442 | 0 |
| 定数倍器 | 0 | 7552 | 636 | 509 | 0 |
| Post-adder | 0 | 10322 | 2862 | 572 | 0 |
| 逆元演算器 | 0 | 9630 | 1197 | 850 | 0 |
| データメモリ | 0 | 0 | 0 | 0 | 9 |
| 回路全体* | 460 | 58191 | 21057 | 4166 | 14 |

* 回路全体でリソース共有があるため合計値と一致しない

表 2　ペアリング計算ハードウェアの最大回路遅延

| 最大回路遅延 [ns] | ロジック遅延 [ns] | 配線遅延 [ns] |
|---|---|---|
| 7.048 | 2.200 | 4.848 |

### 5.2 実装評価結果

表 1 に回路規模の評価結果を示す。DSP は剰余乗算器でのみ使用されており LUT も 30 %以上が剰余乗算器で用いられている。パイプライン型にすることにより、DSP の数はループの段数分だけ設ける必要があり、回路規模が大きくなる。よって、サーバーなどの大量のデータを高速に処理する部分のコプロセッサに向いていると考えられる。Vivado による消費電力解析では剰余乗算器部分で 2W という低消費電力で実装が可能である結果を得た。逆元演算器については LUT9630 個で実装できており、回路全体の 16 %を占めているが、専用回路を設けて削減できるクロック数は全体の 20 %以上であるので、専用回路が高速化に効率的に寄与しているといえる。

表 2 にクリティカルパスディレイの評価結果を示す。剰余乗算器、Post-adder 付近で遅延が大きく、最大回路遅延は 7.048 ns であった。配線遅延が 4.848 ns と大きな割合を占めていることから、ASIC として実装を施した場合の高速化への期待が高いといえる。また、この結果は Vivado のタイミング解析ツールを用いたワースト条件での結果である。実機における最大動作周波数は、実際に FPGA に回路を書き込み、出力が期待値と一致する最大の周波数とし、169 MHz であった。

表 3 に本稿でのペアリングの計算ブロックごとの必要サイクル数とパイプライン空きを示す。パイプライン段数が 18 段の場合、逆元演算を含む $f^{p^6-1}$ と hard part 以外には大きなパイプライン空きが発生しないことがわかる。一方で多少は空きが発生しているのでこれ以上パイプライン段数を大きくすると空きの占める割合が大きくなっていき効率が悪くなることが予想される。

**表 3** ペアリング計算のサイクル数内訳

| 計算ブロック | サイクル数 | パイプライン空き |
|---|---|---|
| MillerLoop | 9528 | 45 |
| Final Addition | 172 | 6 |
| $p^6-1$ 乗 | 546 | 356 |
| $p^2+1$ 乗 | 125 | 7 |
| hard part | 7786 | 189 |
| 計 | 18157 | 603 |

**表 4** ペアリング計算手法の実装比較

| 実装手法 | 回路規模 | 最大周波数 | 演算サイクル数 | 計算時間 [$\mu s$] |
|---|---|---|---|---|
| Ours | 460 DSPs 14548 Slices | 169 MHz | 18157 | 107 |
| 粟野ら 高基数 [12] | 3205k Gates | 147 MHz | 9270 | *62.3 |
| Ghosh ら 高基数 [5] | 144 DSPs 5163 Slices | 166 MHz | 62166 | 375 |
| Han ら 高基数 [8] | 323k Gates | 633 MHz | 330053 | 554 |
| Yao ら 33bitRNS[6] | 32 DSPs 7032 Slices | 250 MHz | 166027 | 916 |
| Yao ら 67bitRNS[7] | 64 DSPs 5237 Slices | 210 MHz | 77769 | 409 |
| Aranha ら SW[2] | Core i5 3570 | 3.4 GHz | 1335000 | 393 |
| Zavattoni ら SW[3] | Core i7 4770 | 3.4 GHz | 1162000 | 341 |

* 配置配線による結果.

表 3 の結果よりペアリング計算一回当たりの計算時間を求め、表 4 に他の実装とともにまとめた。回路規模を他の論文の指標とあわせるために Xilinx の Kintex Ultrascale FPGA の仕様より 4 LUT を 1 Slice と計算し統一している。

比較すると、ソフトウェア実装はハードウェアとさほど差がないほどの高速化を達成している。しかし、汎用 CPU の消費電力は大きく、エネルギー効率を指標にすると数十倍の差が発生する。ハードウェア実装の中では RNS を用いた実装は回路規模が小さくエネルギー効率については優秀である。しかし、基底変換などの計算過程の制約からこれ以上計算時間を短くすることは困難である。高基数実装は回路規模の増大は発生するが短い計算時間を実現している。すなわち、速いレスポンスを必要とするシステムに向いているといえる。我々の実装はこれまで提案されている FPGA 実装の中で最もレイテンシが短い 107 $\mu$s でペアリングが計算できる結果を得た。

## 6. おわりに

本稿では、従来の公開鍵暗号技術より機能性を高めた高機能暗号技術について、その最先端実装の動向及び実装手法についてまとめた。高機能暗号の普及のために、高機能暗号の構成要素であるペアリング計算について汎用 CPU を用いた実装から専用ハードウェアを用いた実装までを比較し、消費電力と処理性能を両立させるためには専用ハードウェアが有用であることを述べた。

また、低レイテンシ実装として、剰余乗算器のループ処理を展開し、さらにスループット向上のためにパイプライン化を行ったパイプライン型剰余乗算器を用いたアーキテクチャについて述べ、ペアリング計算のための最先端ハードウェア実装について示した。

**謝辞** 本研究の一部は，セコム科学技術振興財団の研究助成を受けて行われたものである。同財団の支援に謝意を表する。

### 参考文献

[1] N. Ogura, S. Uchiyama, N. Kanayama, and E.Okamoto,“A note on the pairing computation using normalized Miller functions,” IEICE Trans. on Fundamentals, vol.E95-A, issue.1, pp.196 – 203, 2012.

[2] D. F. Aranha, P. S. L. M. Barreto, P. Longa, and J. E. Ricardini,“The Realm of the Pairings,” in Proc. of Selected Areas in Cryptography (SAC 2013), pp.3 – 25, 2013.

[3] E. Zavattoni, L. J. D. Perez, S. Mitsunari, A. H. Sánchez-Ramírez, T. Teruya, and F. Rodríguez-Henríquez,“Software implementation of an Attribute-Based Encryption scheme,” IEEE Trans on Computers, vol.64, issue.5, pp.1429 – 1441, 2015.

[4] Jean-Luc Beuchat , Jorge Enrique González Díaz , Shigeo Mitsunari, Eiji Okamoto , Francisco Rodríguez-Henríquez, Tadanori Teruya,“High-speed software implementation of the optimal ate pairing over barreto-naehrig curves,” Cryptology ePrint Archive, Report 2010/354, 2010.

[5] S. Ghosh, I. Verbauwhede, and D. Roychowdhur,“Core Based Architecture to Speed Up Optimal Ate Pairing on FPGA Platform,” in Proc. of International Conference on Pairing-Based Cryptography (Pairing 2012), pp.141–159, 2012.

[6] G. X. Yao, J. Fan,R.C.C. Cheung, and I. Verbauwhed,“A High Speed Pairing Coprocessor Using RNS and Lazy Reduction,” Cryptology ePrint Archive, Report 2011/258, 2011.

[7] G. X. Yao, J. Fan, R. C. C. Cheung,and I. Verbauwhede,“Faster Pairing Coprocessor Architecture,” in Proc. of International Conference on Pairing-Based Cryptography (Pairing 2012), pp.160 – 176, 2012.

[8] J. Han, Y. Li, Z. Yu, and X. Zeng,“A 65 nm Cryptographic Processor for High Speed Pairing Computation,” IEEE Trans. on VLSI, vol.23, issue 4, pp.692–701, 2015.

[9] Yang Li Jun Han , Shuai Wang , Dabin Fang ,Xiaoyang Zeng,“An 800MHz cryptographic pairing processor in 65nm CMOS,” in Proc. of Asian Solid-State Circuits Conference (ASSCC 2012), 2012.

[10] H. Orup,“Simplifying quotient determination in high-radix modular multiplication,” in Proc. of 12th IEEE Symposium on Computer Arithmetic, pp.70–77, 1999.

[11] 長浜, 藤本, 松本,“254 ビット素数 BN 曲線上 optimal ate ペアリングの圧縮自乗算による高速計算法の FPGA 実装評価,” 暗号と情報セキュリティシンポジウム (SCIS 2018), 2D4-1, 2018.

[12] 粟野, 市橋, 池田, “2 次拡大体上の汎用演算器を用いた 254bit 素数ペアリング向け ASIC コプロセッサ,” 暗号と情報セキュリティシンポジウム (SCIS 2018), 2D4-3, 2018.