博 士 論 文

3 値論理による公開鍵暗号ハードウェアの研究

指導教官　日比野 靖 教授

主査審査委員　日比野 靖 教授
審 査 委 員　金子 峰雄 教授
審 査 委 員　宮地 充子 助教授
審 査 委 員　田中 清史 助教授
審 査 委 員　高木 剛 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

白勢 政明

2006 年 2 月 1 日

要 旨

公開鍵暗号の主要なタイプに RSA 暗号と ElGamal 暗号とがある．RSA 暗号の暗号化/復号化処理には公開鍵 N を法とする剰余乗算が多数必要であり，ElGamal タイプではある有限体での乗算を繰り返し必要になる．本研究は ElGamal 暗号の一種である XTR に着目している．XTR は従来の ElGamal 暗号と較べて，公開鍵のサイズを 1/3 にでき，暗号化/復号化に必要な処理も効率的になる．本論文では，標数 3 の体を用いると，更に公開鍵のサイズを半分にできることを示す．標数 3 の体での XTR のハードウェア実装が本論文の主題の一つである．

標数 3 の体を実装するために，3 値論理ゲートの構成法である TG 法を提案する．TG 法は現時点では最良と考えられる 3 値論理ゲートの構成法である．TG 法は Olson 法による 1 変数関数ゲートと T ゲートをモジュールとして，任意の関数のゲート構成する方法である．Olson 法だけを用いる方法より，性能や任意の関数のゲートの構成の容易さから，TG 法が優れている．TG 法や Olson 法の長所は，CMOS デバイスと同様に定常時には電流が流れない (つまり消費電力が少ない) ことと，現在の CMOS デバイスの加工技術で製造できることである．実際に 2 値論理のゲートと，TG 法と Olson 法での 3 値論理ゲートの違いは，閾値の異なる MOSFET を複数用意することと複数の電源を用意することだけである．しかし，これらは消費電力の削減のために，2 値論理の CMOS デバイスにも行われるようになっている．

TG 法によるゲートは，従来の CMOS デバイスと構造的にほとんど同じなので，HSPICE でシミュレーションすることができる．本論文で，TG 法による 3 値論理ゲートを用いた，標数 3 の体での XTR のハードウェアを設計し，性能評価を行う．

次に大規模な Wallace tree 乗算器を 3 値論理ゲートを用いて設計することによる利点を議論する．対称 3 進表現を用いると，Wallace tree の構造が著しく単純になり，Wallace tree 乗算器の設計が容易になる．標数が大きな体を用いる ElGamal 暗号と RSA 暗号は，暗号化/復号化の処理に多ビットの乗算が必要であり，3 値論理で設計した大規模乗算器は，これらの暗号の処理に有効である．暗号処理には，通常の乗算の他に，加算や,「Montgomery 乗算」,「擬 Mersenne 素数を法とする剰余計算」というアルゴリズムも用いられるため，これらが対称 3 進表現でも行うことができなければならないが，本論文ではこれらが可能であることを示す．TG 法を用いた 3 値論理ゲートでの，加算器，乗算器，これらのアルゴリズムのためのハードウェアの設計法を具体的に示す．

目 次

図 目 次

表 目 次

第 1 章

緒言

1.1 背景と目的

情報化社会の進展にともない情報セキュリティの重要性が増大している．暗号技術は情報セキュリティにおける主要な技術の 1 つである．1970 年代以前までは，暗号といえば，暗号化のための鍵と復号化のための鍵が同じである共通鍵暗号だけであったが，1976 年に 2 つの鍵が異なっている公開鍵暗号が提案された．公開鍵暗号では，暗号化のための鍵を公開鍵，復号化のための鍵を秘密鍵と呼ぶ．公開鍵暗号では，秘密鍵だけを保管し，公開鍵を広く公開することができる．そのため，大きなネットワークによる不特定多数の人との通信には公開鍵暗号が適している．以前は公開鍵暗号は，暗号化/復号化処理の計算量が膨大なため，実用的ではなかったが，近年の急速な計算機の発達により，十分に実用的になっている．また，公開鍵暗号技術は，電子署名などにも関係している [1].

ところで公開鍵暗号は一般に，従来の共通鍵暗号に対して鍵のサイズが大きく処理のための計算量も多く時間がかかる．XTR[2] と楕円曲線暗号は ElGamal 暗号の一種であり，これらは従来の ElGamal 暗号より公開鍵のサイズを小さくでき，処理も効率的になる．XTR は公開鍵のサイズを 1/3 にでき，処理に必要な計算量は約 1/5 になる．本論文は，標数 3 の体を用いると公開鍵のサイズを更に半分にできることを示し，それをハードウェア実装することが主題の一つとなっている．また，近年楕円曲線暗号に標数 3 の体を用いることが提案されており，そうすることで暗号処理が効率的になる [3, 4, 5].

標数 3 の体は $\{0, 1, 2\}$ 内での演算を主とするため，3 値論理が適している．しかしながら，3 値論理を従来の計算機で実装しようとすると，3 値論理の 1 桁のために 2 ビットが必要となり，無駄が多くなる．そのため本論文では 3 値論理ゲートの設計法についても提案する．多値論理ゲートは以前から研究されているが，これまで研究されてきた多値論理ゲートは，電流の大きさで論理値を決定するか，電圧の大きさで論理値を決定するものでも定常時に電流が流れる仕組みになっ

ており，そのため消費電力が大きくなってしまう．このようなゲートは実用的ではない．現在の計算機に使われている CMOS デバイスは，電圧で論理値を決定し定常時は電流が流れない仕組みになっている．ところで近年 Olson によって，CMOS デバイスと同様に電圧により論理値を決め定常時には電流が流れない多値論理ゲートの構成法が提案された [6]．本論文では，Olson による方法を改良し任意の関数のゲートが容易に構成することができる TG 法を提案する．Olson 法や TG 法では，閾値が異なる複数の MOSFET および電圧の異なる複数の電源を用いること以外は，製造過程は 2 値論理の CMOS デバイスと同じである．また，MOSFET の閾値の調整や複数電圧の使用は，消費電力の削減のために現在ではよく行われている技術となっているので，Olson 法や TG 法は現在の技術でも可能である．将来，より優れた 3 値論理ゲートの構成法が提案されるかも知れないが，TG 法は現時点での最良の方法だと考えられる．

本論文では，TG 法による 3 値論理ゲートを用いて，標数 3 の体での XTR のハードウェアを設計し，性能を評価する．TG 法は構成法がほとんど従来の CMOS デバイスと同じであるので，TG 法によるゲートを HSPICE[7] でシミュレーションすることができる．

次に，3 値論理は標数 3 の体だけではなく，大規模な Wallace tree 乗算器の実装にも適していることを示す．Wallace tree 乗算器は論理段数がビット数の対数に比例する乗算器であり，高い性能が得られる．一般に乗算は加算よりコストがはるかに大きい演算であると言われているが，Wallace tree 乗算器の論理段数は，桁上げ先見加算器の約 2 倍である．必ずしも時間的コストが論理段数だけに依存するわけではないものの，乗算の時間的コストは加算の約 2 倍であると考えることができる．つまり，大規模乗算器を実装できるならば，乗算は加算と較べて時間的コストは著しく大きいわけではない．しかしながら，ビット数が大きくなるにつれ，Wallace tree のノード間の配線が複雑になり，多ビットの Wallace tree 乗算器の実装は困難であった．本論文では，対称 3 進表現を用いると Wallace tree のノード数が約 1/5，配線数が約 1/3.7，配線の交差数が約 1/10 になることを示した．よって，対称 3 進表現を用いると大規模乗算器のための Wallace tree の配置配線が容易になるため，実装が実際的になる．また，対称 3 進表現での桁上げ先見加算加算器も設計できる．対称 3 進表現での演算器は 3 値論理ゲートで実装できる．

RSA 暗号の処理には公開鍵 N(一般に N は 1024 ビット以上) を法とする剰余乗算が多く必要である．また ElGamal 暗号の処理には，用いる体の標数を法とする剰余乗算が多く必要になる．したがって，RSA 暗号と標数の大きな有限体を用いる ElGamal 暗号の処理には，大規模乗算器が有効である．またこれらの暗号処理には，剰余計算を除算を用いないで乗算と加算だけで行うため,「Montgomery 乗算」[8] や「擬 Mersenne 素数を法とする剰余計算」[9] と呼ばれるアルゴリズムも用いられる．これらのアルゴリズムは 2 進表現を用いることを前提としている．本論文では対称 3 進表現でも，Montgomery 乗算と擬 Mersenne 素数を法とする剰余計算の類似があることを示した．これらの暗号に必要な処理はすべて対称 3 進表現で行うことができる．

以上から標数 3 の体での XTR にも，RSA 暗号にも，標数が大きい体を用いる ElGamal 暗号にも 3 値論理が適している．

1.2 論文の構成

本論文は 8 章で構成される．第 2 章では，公開鍵暗号である RSA 暗号と ElGamal 暗号の処理の手順について解説し，ElGamal 暗号で必要となる有限体について説明する．

第 3 章は，ElGamal 暗号の一種である XTR について説明する．XTR は従来の ElGamal 暗号より処理が効率的であり，公開鍵のサイズも 1/3 にすることができる．それから，本論文の中心をなす標数 3 の体を用いた XTR について論ずる．標数 3 の体を用いると，XTR の公開鍵のサイズを更に半分にできることを証明する．第 4 章では，3 値論理ゲートの構成法である Olson 法を説明し，3 値論理ゲートの組織的構成法である TG 法を提案する．また使用する HSPICE のパラメータを決定する．第 5 章では，TG 法による 3 値論理を用いる標数 3 の体での XTR のためのハードウェアを設計し，HSPICE を用いて性能を評価する．

第 6 章では，対称 3 進表現を用いたときの乗算器と加算器についてを輪ずる．乗算器については，Wallace tree の構成法を提案し，配線の複雑さを評価する．対称 3 進表現は乗算器にとってとても有利であることが分かる．加算器については，高性能な桁上げ先見加算器に必要なキャリー生成アルゴリズムについて提案し，アルゴリズムの正当性を証明する．更に TG 法による 3 値論理での乗算器と加算器の設計について議論する．第 7 章では，公開鍵暗号に必要なアルゴリズムである Montgomery 乗算と擬 Mersenne 素数を法とする剰余計算について説明し，これらのアルゴリズムの対称 3 進表現版を提案し，アルゴリズムの正当性を証明する．また，これらのアルゴリズムに必要なハードウェアについて提示する．第 8 章は結言として本論文をまとめる．

第 2章

公開鍵暗号と有限体

現在主流となっている公開鍵暗号にはRSA タイプとElGamal タイプがある．

2.1 有限体と基底

ここでElGamal タイプの暗号プロトコルに必要となる有限体について説明する．

2.1.1 群と体

初めに群と体を定義する．集合 G には演算 $\cdot$ が定義されていて，以下の性質を満たすとき G は群であるという．

1. (結合法則) 任意の $a, b, c \in G$ に対して，$(a \cdot b) \cdot c = a \cdot (b \cdot c)$ が成り立つ．
2. (単位元の存在) 任意の $x \in G$ に対して，$1 \cdot x = x \cdot 1 = x$ となる元 $1 \in G$ がただ1つ存在する．
3. (逆元の存在) 任意の $x \in G$ に対して，$x \cdot x^{-1} = x^{-1} \cdot x = 1$ を満たす $x^{-1} \in G$ が存在する．

例えば，有理数全体の集合 $\mathbb{Q}$ は乗算に関して群であり，整数の全体 $\mathbb{Z}$ は加算に関して群である．

集合 $\mathbb{F}$ には2つの演算 $\cdot$ と $+$ が定義されていて，以下の性質を満たすとき，$\mathbb{F}$ は体であるという．

1. (加法の結合法則) 任意の $a, b, c \in \mathbb{F}$ に対して，$(a + b) + c = a + (b + c)$ が成り立つ．
2. (加法の単位元の存在) 任意の $x \in \mathbb{F}$ に対して，$0 + x = x + 0 = x$ となる元 $0 \in G$ がただ1つ存在する．
3. (加法の逆元の存在) 任意の $x \in \mathbb{F}$ に対して，$x + (-x) = (-x) + x = x$ を満たす $(-x) \in \mathbb{F}$ が存在する．
4. (乗法の結合法則) 任意の $a, b, c \in \mathbb{F}$ に対して，$(a \cdot b) \cdot c = a \cdot (b \cdot c)$ が成り立つ．
5. (乗法の単位元の存在) 任意の $x \in \mathbb{F}$ に対して，$1 \cdot x = x \cdot 1 = x$ となる元 $1 \in \mathbb{F}$ がただ1つ存

在する．

6. (乗法の逆元の存在) 任意の $x \in \mathbb{F}$ に対して，$x \cdot x^{-1} = x^{-1} \cdot x = 1$ を満たす $x^{-1} \in \mathbb{F}$ が存在する．
7. (結合法則) 任意の $a, b, c \in \mathbb{F}$ に対して，$a \cdot (b + c) = a \cdot b + b \cdot c,\ (a + b) \cdot c = a \cdot c + b \cdot c$ が成り立つ．

例えば，有理数全体の集合 $\mathbb{Q}$ や実数全体の集合 $\mathbb{R}$，複素数全体の集合 $\mathbb{C}$ は体であるが，自然数全体の集合 $\mathbb{N}$ は体ではない．

2.1.2 有限体

体である $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ はともに無限集合であるが，有限集合である体 (有限体という) も存在する．$\mathbb{F}$ を有限体とすると，$\#\mathbb{F}$ ($\mathbb{F}$ の元の個数) はある素数のべきになることが知られており，その素数を $\mathbb{F}$ の標数という．元の個数が $q = p^m$ (p は素数) である有限体を $\mathbb{F}_q$ と書く．

$\mathbb{F}_p$ について

p を素数とする．$\mathbb{F}_p$ は集合としては

$$\{0, 1, 2, \cdots, p-1\}$$

であり，加減乗算に対しては普通に計算してから p での剰余をとる．

$$(\mathbb{F}_p \text{での})\ a \pm, \times\ b = a \pm, \times\ b \pmod{p}$$

除算は

$$a \div b = a \times b^{p-2}$$

を計算すればよく，他にはユークリッドのアルゴリズムを使う方法がある．

$\mathbb{F}_q$ について

$m \geq 1$ に対して，$q = p^m$ のとき p を $\mathbb{F}_q$ の標数という．$\mathbb{F}_q$ は $\mathbb{F}_p$ の m 次ベクトル空間とみなすことができ，$\mathbb{F}_q$ の元は $\mathbb{F}_p$ の元を m 個使って表現できる．l が m の約数のとき，$\mathbb{F}_{p^l}$ は $\mathbb{F}_{p^m}$ に含まれていると考えることができる．一般に 2 つの体 K, L に対して，$K \subset L$ のとき K を L の部分体，または L は K の拡大体であるという．さらに，L が K の m 次ベクトル空間になっているときは，L を K の m 次拡大体であるという．$K \subsetneq L$ のとき K を L の真部分体という．

$\mathbb{F}_q$ から $0 = (\underbrace{0, 0, \cdots, 0}_{m\text{ 個}})$ を除いた集合を $\mathbb{F}_q^*$ と書き，$\mathbb{F}_q$ の乗法群と呼ぶ．$\mathbb{F}_q^*$ は乗除算に関して閉じているので，$\mathbb{F}_q^*$ は元の個数が $q-1$ の群である．一般に元の個数が l の有限群 G に対して，すべての $g \in G$ に対して $g^l = 1$ となる．$g \neq 1$ に対して $g^o = 1$ となる 0 より大きい最小の整数 o を g の位数という．o は l の約数になる．$\mathbb{F}_q^*$ は元の個数が $q-1$ の有限群だから $a \in \mathbb{F}_q^*$ に対して，$a^{q-1} = 1$ が成り立つ．

$\mathbb{F}_q^*$ のすべての元は，ある $g \in \mathbb{F}_q^*$ に対して $g^n,\ n \in \mathbb{N}$ の形で書けることが知られている [10, 11]．つまり集合として

$$\mathbb{F}_q^* = \{g^n\ :\ n \in N\}$$

となる．このような g を $\mathbb{F}_q^*$ の生成元 (generator) または原始元 (primitive element) という．$\mathbb{F}_q^*$ のように，すべての元がある元 g のべきで表現されるような群を巡回群という．生成元が g である巡回群を $\langle g \rangle$ と記す．

$\mathbb{F}_q$ の性質

$m \geq 1,\ q = p^m,\ a, b \in \mathbb{F}_q,\ l$ は m の約数とする．このとき $\mathbb{F}_{p^l}$ は $\mathbb{F}_q$ の部分体であり，以下が成り立つ．

$$pa = 0 \tag{2.1}$$

$$(a+b)^p = a^p + b^p \tag{2.2}$$

$$(a+b)^{p^l} = a^{p^l} + b^{p^l} \tag{2.3}$$

$$a^{p^l} = a \Leftrightarrow a \in \mathbb{F}_{p^l} \tag{2.4}$$

$$a^{q-1} = 1 \tag{2.5}$$

詳しくは [10] や [12] を参照．

2.1.3 基底

$q = p^m$ のとき，$\mathbb{F}_q$ は $\mathbb{F}_p$ の m 次ベクトル空間とみなせるので，集合として

$$\mathbb{F}_q = \{a_{m-1}x_{m-1} + a_{m-2}x_{m-2} + \cdots + a_0 x_0\ :\ a_i \in \mathbb{F}_p,\ x_i \in \mathbb{F}_q,\ 0 \leq i \leq m-1\}$$

と書ける．集合 $\{x_{m-1}, x_{m-2}, \cdots, x_0\}$ を $\mathbb{F}_q$ の $\mathbb{F}_p$ 上の基底という．基底の選び方は一般に複数あり，よく用いられる基底のタイプには多項式基底と正規基底がある．$a = a_{m-1}x_{m-1} + a_{m-2}x_{m-2} + \cdots + a_0 x_0$，$b = b_{m-1}x_{m-1} + b_{m-2}x_{m-2} + \cdots + b_0 x_0 \in \mathbb{F}_q$ に対して，$a \pm b$ は，基底の選び方に

依らず，係数同士の加減算によって定義される．

$$\begin{aligned} a \pm b &= (a_{m-1}x_{m-1} + a_{m-2}x_{m-2} + \cdots + a_0 x_0) \pm (b_{m-1}x_{m-1} + b_{m-2}x_{m-2} + \cdots + b_0 x_0) \\ &= (a_{m-1} \pm b_{m-1})x_{m-1} + (a_{m-2} \pm b_{m-2})x_{m-2} + \cdots + (a_0 \pm b_0)x_0 \end{aligned}$$

しかし，どの基底を選ぶかは，乗算や p 乗計算の効率に大きく影響するので，実装上重要な問題となる．基底についての一般的な性質は [11] などを参照.

多項式基底

ここでは多項式基底を説明する．$f(X)$ を $\mathbb{F}_p$ 上既約な $\mathbb{F}_p$ 係数の m 次多項式とする．X のべきの集合 $\{X^{m-1}, X^{m-2}, \cdots, X, 1\}$ は $\mathbb{F}_{p^m}$ の $\mathbb{F}_p$ 上基底となり，このような基底を多項式基底という．$a = a_{m-1}X^{m-1} + a_{m-2}X^{m-2} + \cdots + a_1 X + a_0,\ b = b_{m-1}X^{m-1} + b_{m-2}X^{m-2} + \cdots + b_1 X + b_0 \in \mathbb{F}_{p^m},\ a_i, b_i \in \mathbb{F}_p$ に対して，積 $a \cdot b$ は普通の多項式の積を計算してから，$f(X)$ での剰余をとる．

乗算の計算量は $f(X)$ の形に依存していて，$f(X)$ の多くの項の係数が 0 ならば効率的に乗算ができる．例えば，$f(X) = X^m - 2$ とできるならば，$a \cdot b$ を計算するのに必要な $\mathbb{F}_p$ での乗算は m^2 回である．ただし，2 倍には加算を用いるとする．なお Karatsuba 法 [13] などを使って，更に $\mathbb{F}_p$ での乗算の回数を削減することができる．

このように多項式基底では $f(X)$ をうまく選ぶことにより，$\mathbb{F}_{p^m}$ の乗算を $\mathbb{F}_p$ の乗算 m^2 回で行うことができ，効率的である．ただし，p 乗はバイナリ法を用いると $\log_2 p$ から $2\log_2 p$ 回の $\mathbb{F}_{p^m}$ での乗算をしなければならず，非効率である．

正規基底

$f(X)$ を $\mathbb{F}_p$ 上既約な m 次多項式とする．更に，下の行列式が

$$\begin{vmatrix} X & X^p & X^{p^2} & \dots & X^{p^{m-2}} & X^{p^{m-1}} \\ X^p & X^{p^2} & X^{p^3} & \dots & X^{p^{m-1}} & X \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ X^{p^{m-2}} & X^{p^{m-1}} & X & \dots & X^{p^{m-4}} & X^{p^{m-3}} \\ X^{p^{m-1}} & X & X^p & \dots & X^{p^{m-3}} & X^{p^{m-2}} \end{vmatrix} \neq 0 \bmod f(X)$$

を満たしているとする．このとき $\{X^{p^{m-1}}, X^{p^{m-2}}, \cdots, X^p, X\}$ は $\mathbb{F}_{p^m}$ の $\mathbb{F}_p$ 上基底となり，このような基底を正規基底という．X も $\mathbb{F}_{p^m}$ の元であるから，(2.4) より $X^{p^m} = X$ となる．

正規基底では一般には $\mathbb{F}_{p^m}$ での乗算は複雑だが，p 乗はとても効率的に計算できる．$a = a_{m-1}X^{p^{m-1}} +$

$a_{m-2}X^{p^{m-2}} + \cdots + a_1X^p + a_0X,\ a_i \in \mathbb{F}_p$ とすると

$$\begin{aligned} a^p &= (a_{m-1}X^{p^{m-1}} + a_{m-2}X^{p^{m-2}} + \cdots + a_1X^p + a_0X)^p \\ &= a_{m-1}X^{p^m} + a_{m-2}X^{p^{m-1}} + \cdots + a_1X^{p^2} + a_0X^p \qquad ((2.2)\text{ より}) \\ &= a_{m-2}X^{p^{m-1}} + a_{m-3}X^{p^{m-2}} + \cdots + a_0X^p + a_{m-1}X \quad (X^{p^m} = X\text{ より}) \end{aligned}$$

となる．つまり，p 乗は係数のシフトで行うことができる．

タイプ I の最適化正規基底

多項式基底は乗算は容易だが，p 乗計算は非効率である．これに対して，正規基底は p 乗計算は大変効率的であるが，乗算が難しい．多項式基底と正規基底はともにこのような一長一短を持っている．これら両方の長所を持つ基底に最適化正規基底 (Optimal noamal basis, ONB) がある．なお，p 乗計算は Frobenius 写像に必要であり，Frobenius 写像は有限体 $\mathbb{F}_q$ での計算を効率的に行うためによく用いられている [14]．XTR(3 章や [2] を参照) でも p 乗計算は必要である．

(タイプ I の)ONB とは既約な

$$f(X) = X^m + X^{m-1} + \cdots + X + 1 \tag{2.6}$$

を用いた正規基底であり，多項式基底に類似している性質も持っている．(2.6) は必ずしも既約ではなく，この場合は ONB は存在しない．

$$m+1\text{ が素数} \tag{2.7}$$

$$\text{かつ，}\quad p\text{ が }\mathbb{F}_{m+1}^*\text{ の生成元} \tag{2.8}$$

となるとき $f(X) = X^m + X^{m-1} + \cdots + X + 1$ は $\mathbb{F}_p$ 上の既約多項式になり，さらに集合として

$$\{X^{p^{m-1}},\ X^{p^{m-2}}, \cdots,\ X^p,\ X \pmod{f(X)}\} = \{X^m,\ X^{m-1}, \cdots,\ X^2,\ X \pmod{f(X)}\} \tag{2.9}$$

となる．

p と m は条件 (2.7) と (2.8) を満たしているとすると $X \in \mathbb{F}_{p^m}$ と考えることができ，$\mathbb{F}_{p^m}$ の元として

$$1 = -(X^m + X^{m-1} + \cdots + X^2 + X) \tag{2.10}$$

が成り立つ．また，

$$X^{m+1} - 1 = (X-1)f(X)$$

であるから，

$$X^{m+1} = 1 \tag{2.11}$$

となる．

ONB では $\mathbb{F}_{p^m}$ の元は定数項が 0 である $\mathbb{F}_p$ 係数 m 次多項式で表現され，加減算は多項式の加減算を行い，乗算は多項式の乗算を行ってから (2.10) と (2.11) を用いて $m+1$ 次以上の項と定数項を還元していく．(2.9) より $1 \leq i \leq m$ に対して $X^i = X^{p^j}$ となる $0 \leq j \leq m-1$ があるので，p 乗は係数のシフトでなされる．

タイプ II の最適化正規基底

ONB は乗算にも p 乗計算にも適しているが，(2.7) と (2.8) を満たさなければならず，実際には ONB を用いることができる場合は少ない．そこでタイプ II の ONB が提案されている [15]．

$$2m+1 \text{ が素数} \tag{2.12}$$

$$p \text{ が } \mathbb{F}_{2m+1}^* \text{ の生成元，または } p \text{ が } \mathbb{F}_{2m+1} \text{ で平方剰余 (ただし } \neq 1) \tag{2.13}$$

を満たすとする．剰余に用いる多項式としては

$$f(X) = \frac{X^{2m+1}}{X-1} = X^{2m} + X^{2m-1} + \cdots + X + 1$$

を用いる．(2.12) と (2.13) を満たすとき

$$\{X^m + X^{-m}, X^{m-1} + X^{-(m-1)}, \cdots, X + X^{-1}\}$$

を $\mathbb{F}_{p^m}$ の $\mathbb{F}_p$ 上の基底にすることができる．これをタイプ II の ONB とよぶ．

$X^{-i} = X^{2m+1-i} \bmod f(X)$ であるから，$a \in \mathbb{F}_{p^m}$ に対して，

$$\begin{aligned} a &= a_m(X^m + X^{-m}) + a_{m-1}(X^{m-1} + X^{-(m-1)}) + \cdots + a_1(X + X^{-1}) \\ &= a_1X^{2m} + a_2X^{2m-1} + \cdots + a_{m-1}X^{m+2} + a_mX^{m+1} + a_mX^m + a_{m-1}X^{m-1} + \cdots + a_2X^2 + a_1X \end{aligned}$$

となる．よって，$\mathbb{F}_{p^m}$ での乗算は元を上のような $2m$ 次多項式で表現し，それらを普通に乗算し，$f(X)$ での剰余をとれば良い．この $2m$ 次多項式の係数は対称だから，$\mathbb{F}_{p^m}$ の乗算に必要な $\mathbb{F}_p$ の乗算の回数は m^2 回である．また，タイプ II の ONB でもタイプ I と同様に p 乗は係数のシフトで計算できる．タイプ I とタイプ II を合わせると，ONB を用いることのできる場合が多くなる．

2.2 RSA 暗号

RSA 暗号は素因数分解の困難性を利用した公開鍵暗号である．はじめに RSA 暗号について簡単に説明する．

- **鍵生成**

  2 つの十分に大きな素数 p, q を選び，$N = pq$ を計算する．更にランダムに $1 \leq e \leq (p-1)(q-1)$ を選ぶ．ただし，e と $(p-1)(q-1)$ は互いに素であるとする．さらにユークリッドの拡張アルゴリズムを使って

  $$ed \equiv 1 \pmod{(p-1)(q-1)}$$

  となる $1 \leq d < (p-1)(q-1)$ を計算する．N と e を公開し，d を秘密にする．(N と e が公開鍵，d が秘密鍵．) 素因数分解の困難性から N を素因数分解できないので，p と q を計算できない，つまり $(p-1)(q-1)$ を知ることができない．よって，公開鍵 N と e から秘密鍵 d を求めることはできない．

- **暗号化**

  平文 $0 \leq m \leq N-1$ に対して，

  $$c = m^e \mod N$$

  を計算する．

- **復号化**

  暗号文 $0 \leq c \leq N-1$ に対して，

  $$m' = c^e \mod N$$

  を計算する．

2.3 ElGamal 暗号

ElGamal 暗号は有限体の乗法群の離散対数問題の困難性を利用した公開暗号である．はじめに有限体を素体 $\mathbb{F}_p$ を用いる場合の ElGamal 暗号を説明し，それから一般の有限群による ElGamal 暗号を説明する．

2.3.1 $\mathbb{F}_p$ による ElGamal 暗号

- **鍵生成**

  素数 p と $1 \leq g \leq p-1$ を選ぶ．さらに $1 \leq s \leq p-1$ を選び，$k = g^s \mod p$ を計算する．$g,\ p,\ k$ が公開鍵であり，s が秘密鍵である．なお，p と g は暗号システム参加者共通の情報 (システムパラメータ) とすることができる．離散対数問題の困難性から公開鍵 p と g と $k(= g^s)$ から s を求めることはできない．

- **暗号化**

  乱数 $1 \leq r \leq p-1$ を選ぶ．平文 $0 \leq m \leq p-1$ に対して，

  $$c_0 = g^r \mod p,\ c_1 = m \oplus (k^r \mod p)$$

  を計算する．ここで $\oplus$ はビット毎の排他的論理和をとる演算を表している．$\oplus$ の代わりに普通の乗算を行っても良い．(c_0, c_1) が m の暗号文である．

- **復号化**

  暗号文 (c_0, c_1) に対して

  $$m' = c_1 \oplus (c_0^s \mod p)$$

  を計算すれば良い．(暗号化で $\oplus$ の代わりに乗算を使っている場合は，$m' = c_1/c_0^s \mod p$ を計算する．)

2.3.2 一般的な群での ElGamal 暗号

離散対数問題が困難であるような有限群 G を用いれば，G を使って ElGamal 暗号を構成できる．そのような有限群の例として，有限体 $\mathbb{F}_q$ の乗法群 $\mathbb{F}_q^*$ や楕円曲線の群がある．一般の有限群での ElGamal 暗号は次のようになる．

- **鍵生成**

  $g \in G$ を選ぶ．さらに $1 \leq s \leq \#G$ を選び，$k = g^s \in G$ を計算する．$g,\ p,\ k$ が公開鍵であり，s が秘密鍵である．なお，g は暗号システム参加者共通の情報 (システムパラメータ) とすることができる．離散対数問題の困難性から公開鍵 g と $k(= g^s)$ から s を求めることはできない．

- **暗号化**

  乱数 $1 \leq r \leq \#G$ を選び，G とサイズが同じである平文 m に対して，

  $$c_0 = g^r,\ c_1 = m \oplus k^r$$

  を計算する．ここで g^r, k^r は G の元，$\oplus$ はビット毎の排他的論理和をとる演算を表している．$m \in G$ ならば $\oplus$ の代わりに G の乗算を行っても良い．(c_0, c_1) が m の暗号文である．

- **復号化**

  暗号文 (c_0, c_1) に対して

  $$m' = c_1 \oplus c_0^s$$

  を計算すれば良い．(暗号化で $\oplus$ の代わりに乗算を使っている場合は $m' = c_1/c_0^s$ を計算する．)

2.4 DSA 署名

DSA 署名は離散対数問題に基づく代表的なディジタル署名方式である．ElGamal 暗号のときと同様に，はじめに有限体 $\mathbb{F}_p$ での DSA 署名を説明し，それから一般的な有限群での DSA 署名を説明する．

2.4.1 $\mathbb{F}_p$ による DSA 署名

- **鍵生成**

  $g \in \mathbb{F}_p$ は素数 l に対して $g^l = 1$ となるとする．g をシステムパラメータとする．整数 s をランダムに選んで

  $$k = g^s \pmod{p}$$

  を計算し，k を公開鍵とし s を秘密鍵とする．

- **署名生成**

  文書 (のハッシュ値) m は $0 \leq m \leq l-1$ の範囲にあるとする．

  - $1 \leq r \leq l-1$ の範囲で乱数をとり，

    $$u_1 = g^r \qquad (\mathrm{mod}\ p)$$
    $$u \equiv u_1 \qquad (\mathrm{mod}\ l)$$

    を計算する．
  - 次を計算する．

    $$v \equiv r^{-1}(m + su) \pmod{l} \tag{2.14}$$

    (l は素数だから r^{-1} が求まる．) v は秘密鍵を知らないと生成することはできない．
  - (u, v) が m の署名となる．

- **署名検証**

  1. 署名 (u, v)，文書 m，システムパラメータ g，公開鍵 k から，$u' = (g^{m/v} k^{u/v} \bmod p)$ を計算．
  2. $u' \equiv u \pmod{l}$ ならば署名は正当である．

2.4.2 一般的な群による DSA 署名

G を離散対数問題が困難である有限群とする．

- **鍵生成**

  $g \in G$ の位数は素数 l であるとする．g をシステムパラメータとする．整数 s をランダムに選んで

  $$k = g^s$$

  を計算し，$k \in G$ を公開鍵とし s を秘密鍵とする．

- **署名生成**

  文書 (のハッシュ値) m は $0 \leq m < \#G$ の範囲にあるとする．

  - $1 \leq r \leq l-1$ の範囲で乱数をとり，

    $$\begin{aligned} u_1 &= g^r \\ u &\equiv u_1 \pmod{l} \end{aligned}$$

    を計算する．
  - 次を計算する．

    $$v \equiv r^{-1}(m + su) \pmod{l} \tag{2.15}$$

    (l は素数だから r^{-1} が求まる．) v は秘密鍵を知らないと生成することはできない．
  - (u, v) が m の署名となる．

- **署名検証**

  1. 署名 (u, v)，文書 m，システムパラメータ g，公開鍵 k から，$u' = g^{m/v}k^{u/v}$ を計算．
  2. $u' \equiv u \pmod{l}$ ならば署名は正当である．

2.5 2 章のまとめ

2 章では ONB を用いるときの有限体の計算法と，RSA 暗号と ElGamal 暗号，DSA 署名を説明した．暗号化/復号化には，RSA 暗号では公開鍵の値を法とする剰余乗算，ElGamal 暗号と DSA 署名は用いる有限体の標数を法とする剰余計算を，数多く用いる．ONB を用いると，有限体の乗算，p 乗計算のどちらも効率的ななる．3 章で説明する XTR には乗算も p 乗計算が多数用いられるので，ONB を用いるのが良い．

第 3章

XTR

有限体 $\mathbb{F}_q$ での ElGamal 暗号は $\mathbb{F}_q^*$ の巡回部分群 $G = \langle g \rangle$ の指数計算を用いるが，安全性のため

1. q は 1024 ビット (以上)
2. g の位数は 160 ビット (以上)

となる $\mathbb{F}_q$ と G を用いる．ここで問題となるのは，G が 160 ビットの有限群にも関わらず，G の元を $\mathbb{F}_q$ の元として表現するならば，1024 ビットを要することである．この問題を解消する手段として，トレースを用いる方法があり，トレースを用いる ElGamal 暗号を XTR(Efficient and Compact Subgroup Trace Representation)[2] という．

XTR では，位数が $q^2 - q + 1$ を割る $\mathbb{F}_{q^6}^*$ の部分群 G の元を $\mathbb{F}_{q^2}$ の元として表現できるので，公開鍵のサイズを普通の ElGamal 暗号の 1/3 にできる．

本章は XTR について解説し，XTR においては q を 3 の奇数べきとする (つまり標数 3 の体を用いる) ことで，ある部分群 $G \subset \mathbb{F}_{q^6}$ の元をトレースを用いることで ($\mathbb{F}_{q^2}$ ではなく)$\mathbb{F}_q$ の元で表現できることを示す．これは本研究の成果の一つである．標数 3 の体は従来の計算機では扱いづらいかもしれないが，3 値論理の計算機には適している．また，[3, 4, 5] のように標数 3 の体での楕円曲線暗号の研究も行われている．これらの研究は 3 値論理の実装技術の発展の動機付けとなるかもしれない．

3.1 XTR の概要

q を素数 p のべきとする．有限体 $\mathbb{F}_{q^6}$ の乗法群 $\mathbb{F}_{q^6}^*$ の元の個数は $q^6 - 1$ であり，$q^6 - 1$ は

$$q^6 - 1 = (q+1)(q-1)(q^2+q+1)(q^2-q+1)$$

と因数分解できるので，$h^{q^2-q+1}=1$ となる $h \in \mathbb{F}_{q^6}^*$ が存在する．h は更に

$$h \text{ は } \mathbb{F}_{q^6} \text{の真部分体に含まれない}$$

を仮定する．

$\langle h \rangle$ を用いる離散対数に基づく暗号プロトコルでは，有限体の指数計算 h^n を用いるが，代りに h^n の $\mathbb{F}_{q^2}$ 上トレース

$$Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(h^n) = h^n + h^{nq^2} + h^{nq^4} \in \mathbb{F}_{q^2}$$

を用いる手法が XTR である．ElGamal 暗号と DSA 署名の XTR 版は §3.4 で説明する．

表記を簡単にするために

$$c_n = Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(h^n) = h^n + h^{nq^2} + h^{nq^4}$$

とおく．また，$c = c_1$ とする．XTR では c と n から c_n を計算しなければならないが，次の命題 3.1 や系 3.2 から効率的に計算できる．

命題 3.1

任意の整数 u, v に対して

$$c_{u+v} = c_v c_u - c_v^q c_{u-v} + c_{u-2v} \tag{3.1}$$

が成り立つ．

証明

$h^{q^2-q+1}=1$ であることに注意すると，直接的な計算から (3.1) が成り立つことが分かる． □

系 3.2

c_n に関して次が成り立つ．

$$c_{2n} = c_n^2 - 2c_n^q \tag{3.2}$$

$$c_{n+2} = cc_{n+1} - c^q c_n + c_{n-1} \tag{3.3}$$

$$c_{2n+1} = c_n c_{n+1} - cc_n^q + c_{n-1}^q \tag{3.4}$$

$$c_{2n-1} = c_n c_{n-1} - c^q c_n^q + c_{n+1}^q \tag{3.5}$$

$$c_{pn} = c_n^p \tag{3.6}$$

証明

(3.2) は命題 3.1 で，$u = n+1,\ v = 1$ とおくと得られる．同様に，(3.3) は $u = n+1,\ v = 1$ とおき，(3.4) は $u = n+1,\ v = n$ とおき，(3.5) は $u = n-1,\ v = n$ とおいて，命題 3.1 を用いれば

良い．(3.6) は (2.2) から分かる． □

系 3.2 を使えば，c と $n \in \mathbb{N}$ から c_n を計算できるが，そのためのアルゴリズムは §3.2 で説明する．

3.2 c_n を求めるアルゴリズム

ここでは系 3.2 を使って，c と n から c_n を効率的に計算するアルゴリズムについて説明する．アルゴリズムを記述するのに便利な関数，1 変数関数 $C_{2n}(B)$ と 4 変数関数 $C_{n+2}(A,B,C,D)$, $C_{2n+1}(A,B,C,D)$, $C_{2n-1}(A,B,C,D)$ を次のように定義する．

$$\begin{aligned} C_{2n}(B) &= B^2 - 2B^q \\ C_{n+2}(A,B,C,D) &= C \cdot D - B \cdot D^q + A \\ C_{2n+1}(A,B,C,D) &= B \cdot C - B^q \cdot D + A^q \\ C_{2n-1}(A,B,C,D) &= A \cdot B - B^q \cdot D^q + C^q \end{aligned}$$

すると系 3.2 の各式は次のように書き換えられる．

$$\begin{aligned} c_{2n} &= C_{2n}(c_n) \\ c_{n+2} &= C_{n+2}(c_{n-1}, c_n, c_{n+1}, c) \\ c_{2n+1} &= C_{2n+1}(c_{n-1}, c_n, c_{n+1}, c) \\ c_{2n-1} &= C_{2n-1}(c_{n-1}, c_n, c_{n+1}, c) \end{aligned}$$

ここで

$$S_n = (c_{n-1}, c_n, c_{n+1})$$

とおく．これらの関数を使うと，S_n を求めるアルゴリズムは次のようになる．

アルゴリズム 3.1 [2, Algorithm 2.3.7] (c と $n > 2$ から S_n を求める)

1. 初期値設定

   $C[3] \leftarrow c$

   $C[0] \leftarrow C_{2n}(C[3])$ 　　($\mathbb{F}_q$ の標数が 3 のときは $C[0] \leftarrow C[3]^3$)

   $C[1] \leftarrow C_{2n+1}(3, C[3], C[0], C[3])$

   $C[2] \leftarrow C_{2n}(C[0])$

2. n が偶数ならば n を $n-1$ と取りかえる．

   $n = 2m+1,\ m/2 = \sum_{j=0}^{l} m_j 2^j,\ m_j \in \{0,1\},\ m_l = 1$ とする．

3. for $j = l-1$ down to 0

   $T[1] \leftarrow C_{2n}(C[m_j])$

   $T[2] \leftarrow C_{2n}(C[m_j+1])$

   if $(m_j = 0)$ then $T[3] \leftarrow C_{2n-1}(C[0], C[1], C[2], C[3])$

   if $(m_j = 1)$ then $T[3] \leftarrow C_{2n+1}(C[0], C[1], C[2], C[3])$

   $C[0] \leftarrow T[1]$

   $C[1] \leftarrow T[3]$

   $C[2] \leftarrow T[2]$

4. n が偶数ならば $(C[0], C[1], C[2])$ を返す.

   n が奇数ならば

   $C[0] \leftarrow C_{n+2}(C[0], C[1], C[2], C[3])$

   $(C[1], C[2], C[0])$ を返す.

アルゴリズム 3.1 は n が偶数か奇数でステップ 2 と 4 で必要な操作が異なっている．しかし，c_n だけが必要な場合は，アルゴリズム 3.1 を次のように修正することができる.

アルゴリズム 3.2 (c と n から c_n を求める)

1. 初期値設定

   $C[3] \leftarrow c$

   $C[0] \leftarrow C_{2n}(C[3])$

   $C[1] \leftarrow C_{2n+1}(3, C[3], C[0], C[3])$　　($\mathbb{F}_q$ の標数が 3 のときは $C[1] \leftarrow C[3]^3$)

   $C[2] \leftarrow C_{2n}(C[0])$

2. $n = \sum_{j=0}^{l} n_j 2^j,\ n_j \in \{0,1\},\ n_l = 1$ とする.

3. for $j = l-1$ down to 1

   $T[1] \leftarrow C_{2n}(C[n_j])$

   $T[2] \leftarrow C_{2n}(C[n_j+1])$

   if $(n_j = 0)$ then $T[3] \leftarrow C_{2n-1}(C[0], C[1], C[2], C[3])$

   if $(n_j = 1)$ then $T[3] \leftarrow C_{2n+1}(C[0], C[1], C[2], C[3])$

   $C[0] \leftarrow T[1]$

   $C[1] \leftarrow T[3]$

   $C[2] \leftarrow T[2]$

4. n が偶数ならば $C[0]$ を返す.

   n が奇数ならば $C[1]$ を返す.

3.3 乗算結果のトレース

DSA 署名等の離散対数問題に基づく署名スキームでは，システムパラメータ h と公開鍵 $k = h^s$ と $a, b \in \mathbb{Z}$ から積 $h^a \cdot k^b = h^a \cdot h^{bs} = h^{a+bs}$ を求めることが必要になる．このようなスキームで XTR を用いるならば，システムパラメータ c と公開鍵 c_s と $a, b \in \mathbb{Z}$ から $Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(h^a \cdot g^{bs}) = c_{a+sb}$ を求めなければならない．この計算は公開鍵として c_s の他に c_{s-1} と c_{s+1} も公開されているときには可能である．

アルゴリズム 3.3 $(c, S_y, t < 2^{r+1}$ から $S_{2^{r+1}y+t}$ を求める)

1. $C[3] \leftarrow c,\ C[0] \leftarrow c_{y-1},\ C[1] \leftarrow c_y,\ C[2] \leftarrow c_{y+1}$ とおく．
2. n が偶数ならば n を $n-1$ と取りかえる．
   $2^{r+1} + n = 2m + 1,\ m = \sum_{j=0}^{l} m_j 2^j,\ m_j \in \{0, 1\},\ m_l = 1$ とする．
3. for $j = l$ down to 0
   $T[1] \leftarrow C_{2n}(C[m_j]),\ T[2] \leftarrow C_{2n}(C[m_j + 1])$ とおく．
   $m_j = 0$ ならば，$T[3] \leftarrow C_{2n-1}(C[0], C[1], C[2], C[3])$ を計算．
   $m_j = 1$ ならば，$T[3] \leftarrow C_{2n+1}(C[0], C[1], C[2], C[3])$ を計算．
   $C[0] \leftarrow T[1],\ C[1] \leftarrow T[3],\ C[2] \leftarrow T[2]$ とする．
4. n が奇数ならば，$S_n = (C[0], C[1], C[2])$ となる．
   n が偶数ならば，$C[0] \leftarrow C_{n+2}(C[0], C[1], C[2], C[3])$ とし，$S_n = (C[1], C[2], C[0])$ となる．

アルゴリズム 3.4 $(a, b, c, S_y$ から c_{a+by} を計算)
l を奇素数とし，整数 $0 < a, b, < q$ と S_y および c が与えられているとする．すると c_{a+by} は次のように計算できる．

1. r を $2^r < l < 2^{r+1}$ を満たす整数とする．
2. $d = b/2^{r+1} \bmod l$ と $t = a/d \bmod l$ を計算する．
3. S_y, c からアルゴリズム 3.3 を使って $S_{2^{r+1}y+t}$ を計算する．
4. $S_{2^{r+1}y+t}$ と d からアルゴリズム 3.1 を使って，$c_{d(2^{r+1}y+t)}$ を計算する．
5. $a + by \equiv d(2^{r+1}k + t) \bmod l$ であるから，$c_{a+by} = c_{d(2^{r+1}y+t)}$ である．

アルゴリズム 3.4 はアルゴリズム 3.1 とアルゴリズム 3.3 を 1 回ずつ使って行うことができる．よって DSA 署名のようなスキームにも XTR を適用することはできる．アルゴリズム 3.4 を使って c_{a+sb} を求めるための計算量は h^{a+sb} を求めるための計算量の 1/1.75 である [16]．しかしながら，アルゴリズム 3.4 を使うには c_{s-1}, c_s, c_{s+1} が公開されてなければならず，公開鍵のサイズが 1/3 になるという XTR の長所は失われる．また，より一般的な乗算結果のトレースを求めることはできない．

3.4 XTR を用いた ElGamal 暗号と DSA 署名

アルゴリズム 3.2, 3.3, 3.4 を用いることで，ElGamal 暗号と DSA 署名の XTR 版を構成することができる．$h \in \mathbb{F}_{q^6}$ は $h^{q^2-q+1} = 1$ を満たし，h の位数は素数 l とする．また $c_n = Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(h^n)$ とする．

3.4.1 XTR を用いる ElGamal 暗号

- **鍵生成**

  $1 \leq s \leq l-1$ を選び，$k = c_s$ を計算する．$c,\ k$ が公開鍵であり，s が秘密鍵である．なお，c は暗号システム参加者共通の情報 (システムパラメータ) とすることができる．

- **暗号化**

  乱数 $1 \leq r \leq l-1$ を選ぶ．平文 $0 \leq m \leq l-1$ に対して，

  $$C_0 = c_r,\ C_1 = m \oplus k_r$$

  を計算する．$\oplus$ の代わりに普通の乗算を行っても良い．(C_0, C_1) が m の暗号文である．

- **復号化**

  暗号文 (C_0, C_1) に対して

  $$m' = C_1 \oplus (C_0)_s$$

  を計算すれば良い．(暗号化で $\oplus$ の代わりに乗算を使っている場合は，$m' = C_1/(C_0)_s$ を計算する．)

3.4.2 XTR を用いる DSA 署名

- **鍵生成**

  整数 s をランダムに選んで

  $$c_{s-1},\ c_s,\ c_{s+1}$$

  を計算し，$\{c_{s-1},\ c_s,\ c_{s+1}\}$ を公開鍵とし s を秘密鍵とする．

- **署名生成**

  文書 (のハッシュ値) m は $0 \leq m \leq l-1$ の範囲にあるとする．

  - $1 \leq r \leq l-1$ の範囲で乱数をとり，

    $$\begin{aligned} u_1 &= c_r \\ u &\equiv u_1 \pmod{l} \end{aligned}$$

を計算する.

– 次を計算する.

$$v \equiv r^{-1}(m + su) \pmod{l}$$

– (u, v) が m の署名となる.

- **署名検証**

1. 署名 (u, v), 文書 m, システムパラメータ c, 公開鍵 $\{c_{s-1},\ c_s,\ c_{s+1}\}$ から, $u' = c_{m/v+su/v}$ を計算.
2. $u' \equiv u \pmod{l}$ ならば署名は正当である.

3.5 XTR の計算量

命題 3.3

$\mathbb{F}_{q^2}$ は $\mathbb{F}_q$ 上のタイプ I の ONB を持つか, $q \equiv 3 \pmod 4$ すると, $x, y, z \in \mathbb{F}_{q^2}$ に対して以下が成り立つ.

(i) x^q は $\mathbb{F}_q$ での乗算を必要としない.
(ii) x^2 は 2 回の $\mathbb{F}_q$ 乗算で計算できる.
(iii) $xz - yz^q$ は 4 回の $\mathbb{F}_q$ 乗算で計算できる.

証明

$\mathbb{F}_{q^2}$ がタイプ I の ONB を持つときは [2] を参照. $q \equiv 3 \pmod 4$ のときは $\{1, \sqrt{-1}\}$ が $\mathbb{F}_{q^2}$ の基底になる. これは $\mathbb{F}_{q^2}$ の元 x が, $x_0,\ x_1 \in \mathbb{F}_q$ に対して $x = x_0 + x_1\sqrt{-1}$ と表現されることを意味する.

(i) $q \equiv 3 \pmod 4$ なので

$$(\sqrt{-1})^q = (\sqrt{-1})^3 = -\sqrt{-1}$$

となる. よって, $x = x_0 + x_1\sqrt{-1} \in \mathbb{F}_{q^2}$ に対して

$$\begin{aligned} x^q &= (x_0 + x_1\sqrt{-1})^q \\ &= x_0 + x_1(\sqrt{-1})^q, \quad (2.2),\ (2.3) \text{ より} \\ &= x_0 - x_1\sqrt{-1}, \end{aligned}$$

が成り立つ. よって x^q には乗算を必要としない.

(ii) $x = x_0 + x_1\sqrt{-1} \in \mathbb{F}_{q^2}$ に対して

$$(x_0 + x_1\sqrt{-1})^2 = (x_0 - x_1)(x_0 + x_1) + 2x_0x_1\sqrt{-1}$$

となる．よって，x^2 は 2 回の乗算 $(x_0 - x_1)(x_0 + x_1)$ と $x_0 x_1$ によって得られる．
(iii) $x = x_0 + x_1\sqrt{-1}$, $y = y_0 + y_1\sqrt{-1}$, $z = z_0 + z_1\sqrt{-1} \in \mathbb{F}_{q^2}$ に対して

$$xz - yz^q = (x_0 - y_0)z_0 - (x_1 + y_1)z_1 + ((x_1 - y_1)z_0 + (x_0 + y_0)z_1)\sqrt{-1}$$

が成り立つ．よって，$xz - yz^q$ は 4 回の乗算 $(x_0 - y_0)z_0$, $(x_1 + y_1)z_1$, $(x_1 - y_1)z_0$, $(x_0 + y_0)z_1$ で計算できる． □

命題 3.3 から次が分かる．

系 3.4

$\mathbb{F}_{q^2}$ はタイプ I の $\mathbb{F}_q$ 上 ONB を持つか，$q \equiv 3 \pmod 4$ であるとする．§3.2 で定義した関数 $C_{2n}(B)$, $C_{n+2}(A, B, C, D)$, $C_{2n+1}(A, B, C, D)$, $C_{2n-1}(A, B, C, D)$ に対して，A, B, C, D には $\mathbb{F}_{q^2}$ の元を入れて計算するならば，必要な $\mathbb{F}_q$ 乗算の回数は

C_{2n} は 2 回
C_{n+2} は 4 回
C_{2n+1} は 4 回
C_{2n-1} は 4 回

である．

(証明)
C_{2n} は 1 回の 2 乗計算を行うので，命題 3.3 (ii) より 2 回の $\mathbb{F}_q$ 乗算で計算できる．C_{n+2} は $x = C,\ y = B,\ z = D$ とおくと 1 回の $xz - yz^q$ 計算で計算できるので，命題 3.3 (iii) より 4 回の $\mathbb{F}_q$ 乗算で計算できる．C_{2n+1} は $x = C,\ y = D,\ z = B$ とおき，C_{2n-1} は $x = A,\ y = D^q,\ z = B$ とおくと，それぞれ 1 回の $xz - yz^q$ 計算で計算できる． □

$\mathbb{F}_{q^2}$ は，$\mathbb{F}_q$ 上タイプ I の ONB を持つか，$q \equiv 3 \pmod 4$ であるとする．§3.2 のアルゴリズム 3.1 に必要な $\mathbb{F}_q$ 乗算の回数は，

ステップ 1	8 回 (標数 3 の体のときは 4 回)
ステップ 3 のループ 1 回あたり	8 回

である．よって約 $8 \log_2 n$ 回の $\mathbb{F}_q$ 乗算で，c と n から c_n を計算できる．

3.6 標数 3 の体での XTR

この節では q を標数 3 の体に制限すると，XTR は $\mathbb{F}_{q^6}^*$ のある部分群を $\mathbb{F}_q$ の元 1 つで表現できる，つまり 1/6 のコンパクトさになることを示し，実用的な q を例示する．

3.6.1 標数 3 の体での $\mathbb{F}_q$ 上トレース d_n

$q = 3^{2k-1}$ として，XTR を考える．オリジナルの XTR では，$h^{q^2-q+1} = 1$ となる $h \in \mathbb{F}_{q^6}$ のトレースを用いた．$q = 3^{2k-1}$ のときは更に

$$\begin{aligned} q^2 - q + 1 &= (q + \sqrt{3q} + 1)(q - \sqrt{3q} + 1) \\ &= (q + 3^k + 1)(q - 3^k + 1) \end{aligned}$$

と因数分解できる．よって，$h^{q-\sqrt{3q}+1} = 1$ となる h が存在する[1] ．

このような h に対して

$$h^{q+1} = h^{\sqrt{3q}} \tag{3.7}$$

が成り立つ．(3.7) が標数 3 の体を用いることのキーポイントである．また $\sqrt{3q} = 3^k$ は 3 のべきであるから，(2.3) より $a, b \in \mathbb{F}_{q^6}$ に対して

$$(a+b)^{\sqrt{3q}} = a^{\sqrt{3q}} + b^{\sqrt{3q}} \tag{3.8}$$

が成り立つ．また，$q^3 + 1 = (q+1)(q^2 - q + 1)$ より

$$h^{q^3+1} = 1 \tag{3.9}$$

となる．

d_n を h^n の $\mathbb{F}_q$ 上のトレースと定義する．

$$\begin{aligned} d_n &= Tr_{\mathbb{F}_{q^6}/\mathbb{F}_q}(h^n) \\ &= h^n + (h^n)^q + (h^n)^{q^2} + (h^n)^{q^3} + (h^n)^{q^4} + (h^n)^{q^5} \in \mathbb{F}_q \end{aligned}$$

また $d = d_1$ とする．

次に $c_n = Tr_{\mathbb{F}_{q^6}/\mathbb{F}_{q^2}}(h^n)$ と d_n の関係を調べる．まず，c^q を計算する．

$$\begin{aligned} c^q &= (h + h^{q^2} + h^{q^4})^q \\ &= h^q + h^{q^3} + h^{q^5} \end{aligned}$$

よって，

$$\begin{aligned} c + c^q &= (h + h^{q^2} + h^{q^4}) + (h^q + h^{q^3} + h^{q^5}) \\ &= h + h^q + h^{q^2} + h^{q^3} + h^{q^4} + h^{q^5} \\ &= d \end{aligned}$$

[1] この章のこれ以降，$h^{q-\sqrt{3q}+1} = 1$ を満たす h を扱うが，$h^{q+\sqrt{3q}+1} = 1$ を満たす h でも同様の議論が成り立つのではないかというご指摘を，本大学の宮地充子 助教授から頂きました．この助言は，本論文で提案する標数 3 の体での XTR が有効となる範囲が広がるかもしれないことを意味します．このことについては今後の課題とします．

である．上の式で c を c_n に，h を h^n に置き換えると，

$$d_n = c_n + c_n^q$$

が得られる．よって，c_n から d_n を計算できる．

3.6.2 $\mathbf{d_n}$ から $\mathbf{c_n}$ と $\mathbf{c_n^q}$ を求める方法

今，$q = 3^{2k-1}$ であり，h は $h^{q+1} = h^{\sqrt{3q}}$ を満たしているとする．さらに h は $\mathbb{F}_{q^6}$ の真部分体に含まれないとする．最初に $c \cdot c^q$ を計算する．

$$\begin{aligned}
c \cdot c^q &= (h + h^{q^2} + h^{q^4}) \cdot (h^q + h^{q^3} + h^{q^5}) \\
&= h^{1+q} + h^{1+q^3} + h^{1+q^5} \\
&\quad + h^{q^2+q} + h^{q^2+q^3} + h^{q^2+q^5} \\
&\quad + h^{q^4+q} + h^{q^4+q^3} + h^{q^4+q^5} \\
&= h^{q+1} + h^{q^3+1} + h^{q^6+q^5} && \text{(2.5) より} \\
&\quad + h^{q^2+q} + h^{q^3+q^2} + h^{q^5+q^2} \\
&\quad + h^{q^4+q} + h^{q^4+q^3} + h^{q^5+q^4} \\
&= h^{q+1} + h^{q^3+1} + (h^{q^5})^{q+1} \\
&\quad + (h^q)^{q+1} + (h^{q^2})^{q+1} + (h^{q^3+1})^{q^2} \\
&\quad + (h^{q^3+1})^q + (h^{q^3})^{q+1} + (h^{q^4})^{q+1} \\
&= h^{q+1} + (h^q)^{q+1} + (h^{q^2})^{q+1} + (h^{q^3})^{q+1} + (h^{q^4})^{q+1} + (h^{q^5})^{q+1} \\
&\quad + h^{q^3+1} + (h^{q^3+1})^q + (h^{q^3+1})^{q^2} \\
&= h^{\sqrt{3q}} + (h^q)^{\sqrt{3q}} + (h^{q^2})^{\sqrt{3q}} + (h^{q^3})^{\sqrt{3q}} + (h^{q^4})^{\sqrt{3q}} + (h^{q^5})^{\sqrt{3q}} + 3 && \text{(3.7) と (3.9) より} \\
&= (h + h^q + h^{q^2} + h^{q^3} + h^{q^4} + h^{q^5})^{\sqrt{3q}} && \text{(2.1) と (3.8) より} \\
&= d^{\sqrt{3q}}
\end{aligned}$$

となる．上の式で c を c_n に，h を h^n に置き換えると

$$c_n \cdot c_n^q = d_n^{\sqrt{3q}}$$

が得られる．

2 次方程式の解と係数の関係から，$\mathbb{F}_q$ 係数の 2 次多項式

$$X^2 - d_n X + d_n^{\sqrt{3q}}$$

の根は c_n と c_n^q になる．2 次方程式の解の公式より

$$c_n,\ c_n^q = \frac{d_n \pm \sqrt{d_n^2 - 4d_n^{\sqrt{3q}}}}{2} \tag{3.10}$$

となる．ここで $\mathbb{F}_q$ と c_n の特性をまとめておく．

命題 3.5

$q = 3^{2k-1}$ とし，$h \in \mathbb{F}_{q^6}$ は $h^{q^2-q+1} = 1$ を満たすとする．

(i) $x \in \mathbb{F}_q$ が平方剰余ならば $\sqrt{x} = x^{(q+1)/4}$.

(ii) h の位数は素数 l であるとする．すると，

$$k' \,|\, 6(2k-1) \text{ となるすべての正整数 } k'(< 6(2k-1)) \text{ に対して } l \not| \ 3^{k'} - 1 \tag{3.11}$$

ならば,

(a) $h^n \neq 1$ ならば h^n は $\mathbb{F}_{q^6}$ の真部分体に含まれない,

(b) $1 \leq n \leq l-1$ に対して，$c_n \notin \mathbb{F}_q$,

(c) $N_{\mathbb{F}_{q^6}/\mathbb{F}_{q'}}(h^n) = 1$,

が成り立つ．ここで $N_{\mathbb{F}_{q^6}/\mathbb{F}_{q'}}(h^n)$ は h^n の真部分体 $\mathbb{F}_{q'}$ 上のノルムであり， $A|B$ は A が B を割りきることを意味する．

証明

(i) $q \equiv 3 \pmod 4$ なので，これは [10] で証明されている．

(ii)-(a) $0 \leq n \leq l-1$ に対して

$h^n \in \mathbb{F}_{q^6}$ のある真部分体

$\Rightarrow k'|6(2k-1)$ を満たすある整数 k'に対して $h^{n(3^{k'}-1)} = 1$, (2.4) より

$\Rightarrow l \,|\, n(3^{k'}-1)$,

$\Rightarrow l \,|\, n$, 仮定 (3.11) より

$\Rightarrow n = 0$.

従って，$n \neq 0$ ならば，つまり, $h^n \neq 1$ ならば h^n は $\mathbb{F}_{q^6}$ のどの真部分体にも含まれない.

(ii)-(b) [2] の Lemme 2.2.1 に

$$X^3 - c_n X^2 + c_n^q X - 1 \text{ の根は } h^n,\ h^{nq^2},\ h^{nq^4}.$$

であることが示されている．それで，h^n は 3 次多項式

$$X^3 - c_n X^2 + c_n^q X - 1, \tag{3.12}$$

の根である．$c_n \in \mathbb{F}_q$ ならば (3.12) は $\mathbb{F}_q$ 係数の多項式になる．これは (ii)-(a) の h^n が $\mathbb{F}_{q^6}$ の真部分体に含まれないことに矛盾する．よって， $c_n \notin \mathbb{F}_q$ である．

(ii)-(c) g を $\mathbb{F}_{q^6}$ の原始元とする．すると h の位数は l なので，$0 \leq m < l$ に対して $h = g^{m(q^6-1)/l}$ と書ける．$\mathbb{F}_{q'}$ は $\mathbb{F}_{q^6}$ の真部分体なので，

$$\text{ある整数 } M > 1 \text{ に対して}, \quad q^6 - 1 = M(q' - 1), \tag{3.13}$$

と書ける．$x \in \mathbb{F}_{q^6}$ の $\mathbb{F}_{q'}$ 上ノルム $N_{\mathbb{F}_{q^6}/\mathbb{F}_{q'}}(x)$ は

$$N_{\mathbb{F}_{q^6}/\mathbb{F}_{q'}}(x) = x^{q^6-1/(q'-1)},$$

と定義されるので，

$$\begin{aligned}
N_{\mathbb{F}_{q^6}/\mathbb{F}_{q'}}(h^n) &= N_{\mathbb{F}_{q^6}/\mathbb{F}_{q'}}(g^{nm(q^6-1)/l}), \\
&= (g^{nm(q^6-1)/l})^{(q^6-1)/(q'-1)}, \\
&= g^{nm(q^6-1)^2/(l(q'-1))}. && (3.14) \\
(3.14) \text{ の指数部分} &= \frac{nm(q^6-1)^2}{l(q'-1)}, \\
&= \frac{nmM^2(q'-1)}{l} \qquad (3.13) \text{ より}. && (3.15)
\end{aligned}$$

l は $q^6 - 1 (= M(q' - 1))$ の因数であり，(a) より $l \not| (q' - 1)$ であるから $l | M$ となる．つまり，

$$\text{ある整数 } K \text{ に対して}, \ M = Kl,$$

と書ける．よって，

$$\begin{aligned}
(3.15) &= nmK^2M(q' - 1) \\
&= nmK^2(q^6 - 1) \qquad (3.13) \text{ より} \\
(3.14) &= g^{nmK^2(q^6-1)} \\
&= 1.
\end{aligned}$$

□

R を (3.10) の根号の中の値とする．命題 3.5 (ii) より $\sqrt{R} \notin \mathbb{F}_q$ であるが，$\sqrt{-1} \notin \mathbb{F}_q$ であるので，$\sqrt{-R} \in \mathbb{F}_q$ であることは簡単に分かる．よって命題 3.5 (i) を使って $\sqrt{-R}$ を計算できる．式 (3.10) は次のように書き換えることができる．

$$\begin{aligned}
c_n, c_n^q &= \frac{d_n \pm \sqrt{R}}{2} \\
&= 2(d_n \pm \sqrt{-R} \cdot \sqrt{-1}) \\
&= 2d_n \pm (2d_n^2 + d_n^{\sqrt{3q}})^{(q+1)/4} \cdot \sqrt{-1}. && (3.16)
\end{aligned}$$

3.6.3 d_n の求め方

d が与えられたら c と c^q を求められることが分かった．$\{c, c^q\}$ から c を選べれば，c_n を求めそれから $d_n = c_n + c_n^q$ を計算できる．c^q を選ぶとどうなるかを説明する．系 3.2 の (3.6) より $c^q = c_q$ であり，c_q と n から $c_{qn} = c_n^q$ を計算できる．初めに $c_q + c_q^q = d$ であることを確認する．

$$\begin{aligned} c_q + c_q^q &= c^q + c^{q^2} && (3.6) \text{ より} \\ &= (h + h^{q^2} + h^{q^4})^q + (h + h^{q^2} + h^{q^4})^{q^2} && c \text{ の定義より} \\ &= (h^q + h^{q^3} + h^{q^5}) + (h^{q^2} + h^{q^4} + h^{q^6}) && (2.3) \text{ より} \\ &= h + h^q + h^{q^2} + h^{q^3} + h^{q^4} + h^{q^5} && (2.5) \text{ より} \\ &= d && d \text{ の定義より} \end{aligned}$$

上の h を h^n に，c を c_n に置き換えれば，

$$d_n = c_{qn} + c_{qn}^q$$

となる．よって，$c^q = c_q$ と n から c_{qn} を計算し，それから d_n が求められる．

以上から，d と n が与えられたら d_n を計算できる．

定理 3.6

与えられた d と n が与えられたら，次のようにして d_n を求めることができる．

(i) (3.16) を使って d から c と c^q を求め，ランダムに 1 つを選びそれを c' とする．
(ii) アルゴリズム 3.1 を使って c' と n から c'_n を求める．
(iii) $d_n = c'_n + c'^q_n$ を計算する．

d_n を求めるための計算は c_n を求めるときと較べて (i) と (iii) が増えた．

3.6.4 標数 3 の体での XTR を用いる ElGamal 暗号

d と n から d_n を計算できるので，標数 3 の体での XTR を用いる ElGamal 暗号を構成できる．$h \in \mathbb{F}_{q^6}$ は $h^{q-\sqrt{3q}+1} = 1$ を満たし，h の位数は素数 l とする．また $d_n = Tr_{\mathbb{F}_{q^6}/\mathbb{F}_q}(h^n)$ とする．

- **鍵生成**

  $1 \le s \le l-1$ を選び，$k = d_s$ を計算する．d, k が公開鍵であり，s が秘密鍵である．なお，d は暗号システム参加者共通の情報 (システムパラメータ) とすることができる．

- **暗号化**

  乱数 $1 \le r \le l-1$ を選ぶ．平文 $0 \le m \le l-1$ に対して，

$$C_0 = d_r,\ C_1 = m \oplus k_r$$

を計算する．$\oplus$ の代わりに普通の乗算を行っても良い．(C_0, C_1) が m の暗号文である．

- **復号化**

  暗号文 (C_0, C_1) に対して

  $$m' = C_1 \oplus (C_0)_s$$

  を計算すれば良い．(暗号化で $\oplus$ の代わりに乗算を使っている場合は，$m' = C_1/(C_0)_s$ を計算する．)

なお，本論文では d_{a+bk} の計算法を提示できなかった．従って，標数 3 の体での XTR を用いる DSA 署名は，現時点では構成できない．これは今後の課題としたい．

3.7 標数 3 の体の選び方

この節では，$101 \leq 2k-1 \leq 199$ の範囲で，体 $\mathbb{F}_q$, $q = 3^{2k-1}$ の選び方を議論する．XTR で用いる $\mathbb{F}_q$ と $d = Tr_{\mathbb{F}_{q^6}/\mathbb{F}_q}(h)(h \in \mathbb{F}_{q^6})$ を選ぶときは，安全性を考慮しなければならない．これまでと同様に $h^{q-\sqrt{3q}+1} = 1$ を満たすとする．

オリジナルの XTR と本論文で提案した標数 3 の体での XTR の安全性は $\langle h \rangle$ での離散対数問題 (DLP) の困難性に依存している．数体篩法は有限体での DLP を解くためのよく知られたアルゴリズムであり，その計算時間は $L[p^t, 1/3, 1.923]$ であると予想されている[2]．ここで $L[n, v, u] = \exp((u + o(1))(\ln n)^v(\ln \ln n)^{1-v})$，$\mathbb{F}_{p^t}$ は $\langle h \rangle$ を含む最小の体である．

h が $\mathbb{F}_{q^6}$ の真部分体に含まれないならば，$\langle h \rangle$ での DLP を解くための計算時間は $L[q^6, 1/3, 1.923]$ となるが，h が $\mathbb{F}_{q^6}$ の真部分体 $\mathbb{F}_{3^{k'}}\,(k'|6(2k-1))$ に含まれるならば，その計算時間は $L[3^{k'}, 1/3, 1.923]$ となり，XTR の安全性レベルを下げることになる．

Pollard の ρ アルゴリズムは任意の群での DLP を解くためのアルゴリズムであり，$\langle h \rangle$ での DLP を解くために用いることができる．l を h の位数の最大素因数とすると，このアルゴリズムの計算時間は $O(\sqrt{l})$ である．

一般に $\lceil \log_2 p^t \rceil = 1024$ と $\lceil \log_2 l \rceil = 160$, あるいは $\lceil \log_2 p^t \rceil = 3072$ と $\lceil \log_2 l \rceil = 256$ ならば，数体篩法による $\mathbb{F}_{p^t}$ での DLP と，Pollard の ρ 法による $\langle h \rangle$ での DLP の困難性が同じであると言われている [17].

$$\exp((1.923 - 0.15)(\ln 2^{1024})^{1/3}(\ln \ln 2^{1024})^{2/3}) \approx \sqrt{2^{160}},$$
$$\exp((1.923 - 0.15)(\ln 2^{3072})^{1/3}(\ln \ln 2^{3072})^{2/3}) \approx \sqrt{2^{256}}.$$

[2] p の値が小さいときは L の中の 1.923 は小さくなる傾向がある．特に $p = 2$ とき 1.53 となる．$p = 3$ ではどうなるか分からないが，本論文の値より大きい q を使うことが必要になるかもしれない．

となるので，o は整数であるとみなすと $o=-0.15$ となる．$\sqrt{l}=L[p^t,1.923,1/3]$ ならば

$$\lceil \log_2 l \rceil = \lceil 2\log_2 L[p^t,1.923,1/3] \rceil,$$

となる．関数 $L'(x)$ を $o=-0.15$ として

$$L'(x) = \lceil 2\log_2 L[x,\ 1/3,\ 1.923] \rceil,$$

と定義する．すると $L'(q^6) < \lceil \log_2 l \rceil$ ならば，Pollard の ρ 法による $\langle h \rangle$ での DLP は，数体篩法による $\mathbb{F}_{q^6}$ での DLP より困難であると見なすことができる．

XTR はこれまで，$c_n \in \mathbb{F}_{p^2}$ や $h \in \mathbb{F}_{p^6}$ [2]，または $c_n \in \mathbb{F}_{p^{2k}}$ や $h \in \mathbb{F}_{p^{6k}}$ [18] の場合のみが扱われてきた．ここで p と k は素数である．これに対してこの論文では $c_n \in \mathbb{F}_{3^{2(2k-1)}}$ や $h \in \mathbb{F}_{3^{6(2k-1)}}$ となる場合を扱っており，$2k-1$ は合成数になるかもしれない．従って，$\mathbb{F}_{q^6}$ に多くの部分体が存在することがあり得，ノルムはべき乗の指数の情報の一部を保存するので，この論文で扱う XTR に対してはノルム攻撃に注意しなければならない．但し，すべての真部分体 $\mathbb{F}_{q'}$ に対して，$N_{\mathbb{F}_{q^6}/\mathbb{F}'}(h^n)=1$ となるならば，ノルム攻撃は効力を持たない．

以上から XTR で用いる標数 3 の体は次の条件を満たせば良い．

条件

1. h は $\mathbb{F}_{q^6}$ のどの真部分体にも含まれない，

2. $L'(q^6) \le \lceil \log_2 l \rceil$，

3. $\mathbb{F}_{q^6}$ のどの真部分体 $\mathbb{F}_{q'}$ に対しても $N_{\mathbb{F}_{q^6}/\mathbb{F}_{q'}}(h^n)=1$.

h の位数が素数ならば，この値は l であり，l は $q-\sqrt{3q}+1$ の最大素因数である．すべての $q=3^{2k-1}$ と $101 \le 2k-1 \le 199$ と $k'|6(2k-1)$ を満たす k' に対して，$\gcd(q-\sqrt{3q}+1,\ 3^{k'}-1) < l$ となることを確かめることができる．これは，すべての $2k-1$ が命題 3.5 の条件 (3.11) “すべての k' に対して，$l \not| 3^{k'}-1$” を満たすことを意味する．従って，$h(\neq 1)$ の位数が l ならば，命題 3.5-(ii) より**条件 1** と**条件 3** を満たすことが分かる．

条件 2 が満たされるかどうかは表 3.1 より分かる．**条件 1, 2, 3** すべてを満たす良い $2k-1$ は

$$\begin{aligned} 2k-1 \quad = \quad & \mathbf{141}, 149, 153, 163, 167, 169, 177, \\ & 183, 187, \mathbf{189}, 193, 197, \end{aligned}$$

である．なお更に $\mathbb{F}_{3^{2k-1}}$ が $\mathbb{F}_3$ 上のタイプ II の ONB を持つときは太字となっている．なお，(2.7) より $\mathbb{F}_{3^{2k-1}}$ は $\mathbb{F}_3$ 上のタイプ I の ONB を持つことはあり得ない．$2k-1=107$ が現在の標準的な安全性を与えるような大きさ (1024 ビット) の有限体に対応するが，良い $2k-1$ はこの周辺の値を含んでいない．そのため，**条件 2** を少し緩める．

条件

2'. $0.95L'(q^6) < \lceil \log_2 l \rceil (\leq L'(q^6))$.

このような $2k-1$ は XTR のコンパクトさを少し失わせるが，それでもかなり良い値である．かなり良い $2k-1$ は

$$2k-1 = \mathbf{111}, 157, \mathbf{165}, \mathbf{179}, 181, 199,$$

であり，太字は $\mathbb{F}_{3^{2k-1}}$ が $\mathbb{F}_3$ 上タイプ II の ONB を持つことを示している．なお，表 3.1 の素因数の大きさは，PARI/GP [19] を使って計算した．

$\mathbb{F}_q$ が $\mathbb{F}_3$ 上 ONB を持つことは XTR の計算の効率性を与えるが，三項多項式基底などを用いても良いかもしれない．

表 3.1: $q = 3^k$ の選択について

$2k-1$	$\lceil \log_2 q^6 \rceil$	$L'(q^6)$	$\lceil \log_2 l \rceil$	(∗1)	(∗2)	$2k-1$	$\lceil \log_2 q^6 \rceil$	$L'(q^6)$	$\lceil \log_2 l \rceil$	(∗1)	(∗2)
101	961	156	115	3	持たない	151	1436	185	104	3	持たない
103	980	157	142	3	持たない	153	1456	186	199	15	持たない
105	999	158	60	34	持つ	155	1475	187	116	50	持つ
107	1018	160	137	1	持たない	157	1494	188	180	1	持たない
109	1037	161	71	1	持たない	159	1513	189	87	6	持たない
111	1056	162	156	5	持つ	161	1532	190	124	37	持たない
113	1075	163	70	3	持つ	163	1551	191	256	3	持たない
115	1094	165	120	37	持たない	165	1570	192	191	53	持つ
117	1113	166	129	15	持たない	167	1589	193	237	1	持たない
119	1132	167	111	27	持つ	169	1608	194	218	21	持たない
121	1151	168	62	18	持たない	171	1627	195	158	15	持たない
123	1170	170	61	6	持たない	173	1646	196	145	3	持つ
125	1189	171	85	40	持つ	175	1665	197	70	56	持たない
127	1208	172	80	3	持たない	177	1684	198	233	5	持たない
129	1227	173	84	6	持たない	179	1703	199	194	1	持つ
131	1246	174	122	1	持つ	181	1722	200	198	1	持たない
133	1265	175	104	31	持たない	183	1741	201	234	5	持たない
135	1284	176	75	43	持たない	185	1760	202	127	59	持たない
137	1303	178	155	3	持たない	187	1779	203	245	27	持たない
139	1322	179	157	3	持たない	189	1798	204	233	43	持つ
141	1341	180	193	5	持つ	191	1817	205	140	1	持つ
143	1360	181	140	21	持たない	193	1836	206	306	1	持たない
145	1379	182	104	46	持たない	195	1855	207	157	63	持たない
147	1398	183	129	34	持たない	197	1874	208	209	3	持たない
149	1417	184	220	3	持たない	199	1893	208	207	3	持たない

(∗1): $k' | 6(2k-1),\ k' < 6(2k-1)$ に対する $\gcd(q - \sqrt{3q} + 1, 3^{k'} - 1)$ の最大値のビット数
(∗2): $\mathbb{F}_q$ が $\mathbb{F}_3$ 上タイプ II の ONB を持つかどうか

3.8　3 章のまとめ

3 章では XTR について説明した．同じ安全性に対して，XTR は ElGamal 暗号の暗号化/復号化の計算を効率にし，公開鍵のサイズも 1/3 になる．標数 3 の体を用いると，XTR の公開鍵を更に半分のサイズにできることを示した．これは本研究の成果の 1 つである．また， 実用的な標数 3 の体の選び方も議論した．

第 4 章

3 値論理ゲート

本章では Olson 法 [6] による 3 値論理ゲートの構成法を説明し，TG(トランスファーゲート) 法 [20, 21] を提案する．Olson 法は閾値の異なる NMOSFET と PMOSFET を用いることで，多値論理ゲートを構成する方法である．[6] では「閾値 0.5V の NMOSFET」などの表現によって書かれているが，本論文は HSPICE [7] レベル 49 のパラメータを例示することで，より実用に近づいている．TG 法は Olson 法による 1 変数関数ゲートとトランスファーゲートとを組み合わせて 2 変数関数ゲートを構成する方法であり，本論文の著者は提案者の一人である．HSPICE のシミュレーションにより，[6] で述べられている閾値の種類より若干種類を増やすと，ゲートの性能が格段に向上することが分かった.

4.1 3 値論理

3 値論理ゲートの構成法の前に 3 値論理全般について簡単に説明する．[22] の序論に書かれているように，多値論理は，高速性，ピン数の削減，配線量の削減，トランジスタなどの素子数の削減，インターフェースの容易性などの理由から研究がなされてきた.

4.1.1 3 値論理の基本関数

L を論理値をとる集合とする．3 値論理では $L = \{0, 1, 2\}$ を用いることが一般的だが，対称 3 進表現を考えるときに便利なので，本論文では

$$L = \{-1, 0, 1\}$$

とする．本節では 6 つの関数 OR，AND，NOT，ADD，PRO，CAR を定義する．ここで OR と AND は [22] で定義されている関数であり，NOT は反転として定義されている．$x, y \in L$ に対し

て，各関数は次のように定義する．

$$\begin{aligned}
\mathrm{OR}(x,y) &= min\{x,y\} \\
\mathrm{AND}(x,y) &= max\{x,y\} \\
\mathrm{NOT}(x) &= -x \bmod 3 \\
\mathrm{ADD}(x,y) &= x+y \bmod 3 \\
\mathrm{PRO}(x,y) &= x\cdot y \bmod 3 \\
\mathrm{CAR}(x,y) &= \begin{cases} 1 & x=1, y=1 \text{ のとき} \\ -1 & x=-1, y=-1 \text{ のとき} \\ 0 & \text{その他の場合} \end{cases}
\end{aligned}$$

[22] では $\mathrm{OR}(x,y)$ を $x+y$，$\mathrm{AND}(x,y)$ を $x\cdot y$ と書いており，2 値論理でも一般に OR を $+$，AND を $\cdot$ と表記することが多い．しかしながら，本論文は主に対称 3 進表現や標数 3 の体での演算器を扱っているので，本論文では

$$\begin{aligned}
x+y &= \mathrm{ADD}(x,y) \\
x\cdot y &= \mathrm{PRO}(x,y)
\end{aligned}$$

と表記することにする．CAR は対称 3 進表現での 2 つの 1 桁の値の加算を行ったときの桁上げ (キャリー) である．CAR は演算器を考えるに際して重要になり，今後良く出てくるので，CAR にも演算子を用いることにする．

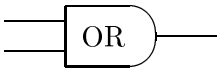
$$\mathrm{CAR}(x,y) = x \odot y$$

CAR に $\odot$ を用いることにしたのは，演算器を考える場合，CAR が 2 進表現の AND と性質が似ているからである．

$\overline{x}$ に関しては，一般的な書き方と同じように

$$\overline{x} = \mathrm{NOT}(x)$$

とする．ここで定義した関数のゲートを



などと図示することにする．

4.1.2 T 演算子と縮退 T 演算子

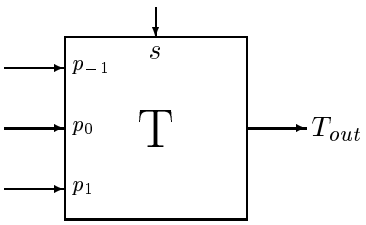
R 値 T 演算子は

$$T(p_0, p_1, \cdots p_{R-1}\ ;\ s) = p_i \quad (s = i \text{ のとき})$$

で定義される．$L=\{-1,0,1\}$ での 3 値論理では

$$T(p_{-1},p_0,p_1\ ;\ s)=p_i\quad (s=i\ \text{のとき})$$

と定義される．T 演算子を実現するためのゲートを T ゲートと呼ぶ．



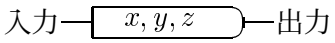
T 演算子で 2 つの制御信号 i,j で同じ値が選択されるような縮退された T 演算子を T_{ij} と書くことにする．制御信号 i がない縮退された T 演算子を $\mathrm{T}_{\overline{i}}$ と書くことにする．これらの縮退された T 演算子を縮退 T 演算子と呼ぶことにする．3 値論理では縮退 T 演算子は T_{-10}，T_{-11}，T_{01}，$\mathrm{T}_{\overline{-1}}$，$\mathrm{T}_{\overline{0}}$，$\mathrm{T}_{\overline{1}}$ の 6 種類がある．

4.1.3　一般的な関数

$x,y,z\in\{-1,0,1\}$ に対して，引数が -1 のときに x を，引数が 0 のときに y を，引数が 1 のときに z を返す 1 変数関数を

$$(x,y,z)$$

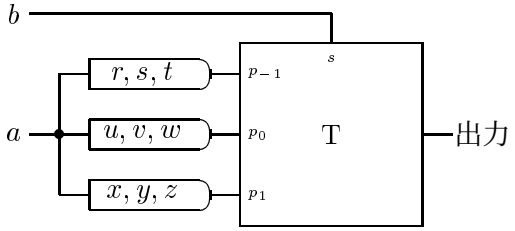
と表記することにする．また (x,y,z) を実現するための 3 値論理ゲートを

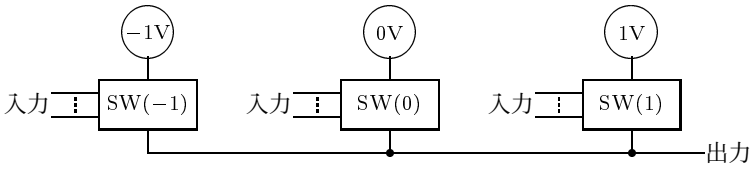


と図示することにする．すると，カルノー図

		a		
		-1	0	1
	-1	r	s	t
b	0	u	v	w
	1	x	y	z

で定義される 2 変数関数のゲートは 1 変数関数 (r,s,t)，(u,v,w)，(x,y,z) のゲートと T ゲートから構成できる．





SW(−1)，SW(0)，SW(1) のうち 1 つだけがオンとなる．

図 4.1: 3 値ゲートの概要図

このように 3 値論理の 1 変数関数ゲートと T ゲートから，任意の 2 変数関数ゲートを構成できる．同様の方法により，3 値論理のより大きな多変数関数ゲートも構成できる．つまり，3 値論理ゲートを作成するためには，1 変数ゲートと T ゲートだけを精密に構成すれば良い．3 値論理の 1 変数関数は $3^3 = 27$ 種類あり，これは 2 値論理の 1 変数関数に比べるとかなり多いが，各関数のゲートを吟味できない程には多くない．しかも 4 種類 $(-1,-1,-1),(0,0,0),(1,1,1),(-1,0,1)$ はゲートを必要としない関数である．

4.2 Olson 法による 3 値論理ゲート

Olson によって提案された多値論理ゲート構成法 [6] により，任意の 3 値論理関数ゲートを構成することができるが，3 値論理の 2 変数関数は $3^{3^2} = 19,683$ 種類あり，各関数のためのゲートをすべて用意することは事実上不可能である．また，関数によってはゲート内のトランジスタの接続が複雑になり，遅延時間が大きくなる．前節で示したように，1 変数関数ゲートと T ゲートを組み合わせることで，任意の関数のゲートを構成できる．本論文では，3 値論理ゲートの構成法として，Olson 法による 1 変数関数ゲートとトランスファーゲートを用いる T ゲートとを組み合わせる方法 (TG 法，トランスファーゲート法) を提案する．

初めに，論理値 $\{-1,0,1\}$ を用いる 3 値論理のためのゲートの構成法を説明する．また，論理値 -1 には -1V，論理値 0 には 0V，論理値 1 には 1V の電圧が対応しているとする．

Olson が提案した構成法による 3 値論理ゲートは，図 4.1 のように 3 値に対応した 3 つのソース電源と 3 つのスイッチ SW(-1)，SW(0)，SW(1) から構成され，どのような入力に対しても，3 つのスイッチのうち 1 つだけがオンとなり残りの 2 つはオフとなるようにする．

各スイッチ SW(n), $n \in \{-1,0,1\}$ はいくつかの NMOSFET と PMOSFET を直列/並列に接続することで構成される．ゲートを構成したい関数が，どんな入力に対しても n を返さないならば，スイッチ SW(n) は必要ない．

各 SW(n) を構成するためには，次の I から VI までの 6 種類の動作をするスイッチが必要である．

入力	I	II	III	IV	V	VI
-1	off	off	on	on	on	off
0	off	on	on	off	off	on
1	on	on	off	off	on	off

このうち V は I と IV を並列に接続することで実現でき，VI は II と III を並列に接続することで実現できる．よって I から IV までのスイッチがあれば，SW(n) の構成には十分である．

Olson 法では，3 値ゲートの構成には閾値の異なる NMOSFET を 3 種類，PMOSFET も閾値の異なる 3 種類を用いて，図 4.1 の各スイッチ SW(n) を構成する．後で述べるように，もう少し MOSFET の種類を増やすと，ゲートの性能が良くなる．なお，ゲートの構成のために閾値の異なる MOSFET や複数の電源電圧を用いることは，消費電力を削減のために今日では一般的に行われている．

各 MOSFET を表 4.1 のように名前付けする．N や n は NMOSFET を表し，P や p は PMOSFET を表している．E や e はエンハンスメント型を，d はディプリーション型を表している．大文字は小文字より閾値の絶対値が大きいことを意味している．

また，NMOSFET のバックバイアスは閾値に関係なく -1V の電源電圧に接続し，PMOSFET のバックバイアスは 1V の電源電圧に接続する．

4.2.1 ソース電源が -1 V のスイッチで使用する MOSFET

MOSFET の性質上，SW(-1) には NMOSFET しか使用できない．MOSFET への入力が -1 V のときはゲート・ソース間電圧 V_{gs} は 0 V，入力が 0 V のときは V_{gs} は 1 V，入力が 1 V のときは V_{gs} は 2 V となり，NE と ne のスイッチ動作は

表 4.1: MOSFET の名称と種類，閾値

MOSFET 名	閾値と MOSFET の種類
NE	閾値が 1V と 2V の間 (例えば 1.5V) のエンハンスメント型 NMOSFET
ne	閾値が 0V と 1V の間 (例えば 0.5V) のエンハンスメント型 NMOSFET
nd	閾値が -1V と 0V の間 (例えば -0.5V) のディプリーション型 NMOSFET
PE	閾値が -2V と -1V の間 (例えば -1.5V) のエンハンスメント型 PMOSFET
pe	閾値が -1V と 0V の間 (例えば -0.5V) のエンハンスメント型 PMOSFET
pd	閾値が 0V と 1V の間 (例えば 0.5V) のディプリーション型 PMOSFET

入力	V_{gs} (V)	NE (I)	ne (II)
-1	0	off	off
0	1	off	on
1	2	on	on

となる．単独の MOSFET では III と IV は実現できない．

4.2.2 ソース電源が 0 V のスイッチで使用する MOSFET

SW(0) では NMOSFET，PMOSFET の両方を使用できる．MOSFET への入力が -1 V のときは V_{gs} は -1 V，入力が 0 V のときは V_{gs} は 0 V，入力が 1 V のときは V_{gs} は 1 V であるので，各 MOSFET の動作は次のようになる．

入力	V_{gs} (V)	ne (I)	nd (II)	pd (III)	pe (IV)
-1	-1	off	off	on	on
0	0	off	on	on	off
1	1	on	on	off	off

4.2.3 ソース電源が 1 V のスイッチで使用する MOSFET

SW(1) では PMOSFET のみを使用できる．MOSFET への入力が -1 V のときは V_{gs} は -2 V，入力が 0 V のときは V_{gs} は -1 V，入力が 1 V のときは V_{gs} は 0 V であるので，各 MOSFET の動作は

入力	V_{gs}(V)	pe (III)	PE (IV)
-1	-2	on	on
0	-1	on	off
1	0	off	off

となる．単独の MOSFET では I と II は実現できない．

4.2.4 SW(n) の構成法

各 SW(n) の構成には，I から VI の動作ができるスイッチが必要である．SW(-1) に関してはスイッチは I と II しか用意されていないが，$(1, 1, -1)$ ゲートの出力を NE か ne の入力とすると，III の動作になる．NE と ne のどちらを選ぶべきかは，リーク電流やゲートの性能を考慮して決めれば良い．同様に SW(-1) では，$(1, -1, -1)$ ゲートの出力を NE か ne の入力とすると，IV の動作になる．

SW(1) に関してもスイッチは III と IV しかないが，SW(-1) のときと同様に $(1, 1, -1)$ ゲート，$(1, -1, -1)$ ゲート，PE，pe から I と II を実現できる．

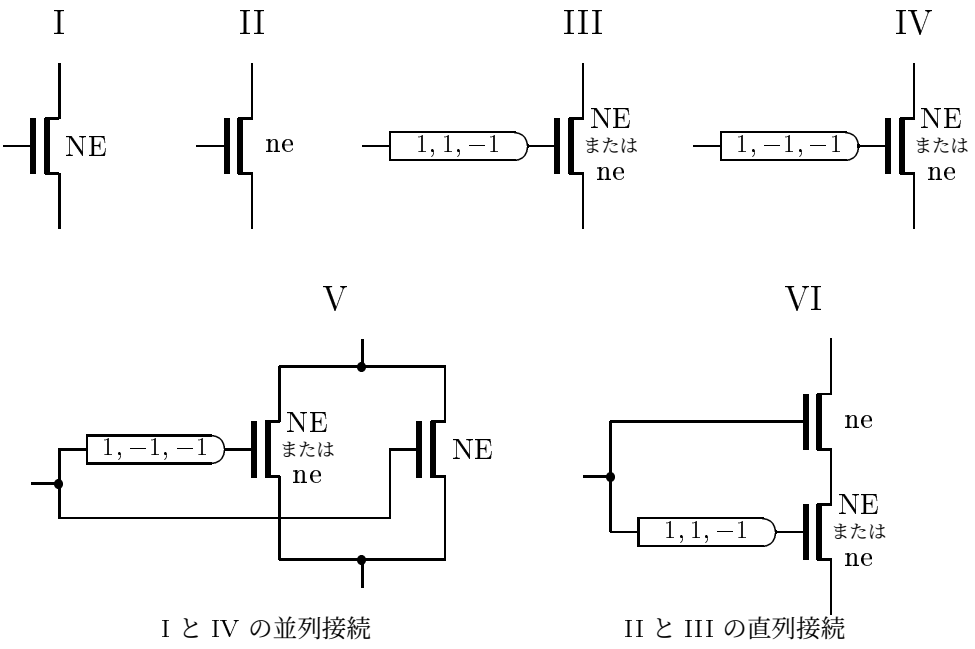


図 4.2: SW(−1) で用いるスイッチ

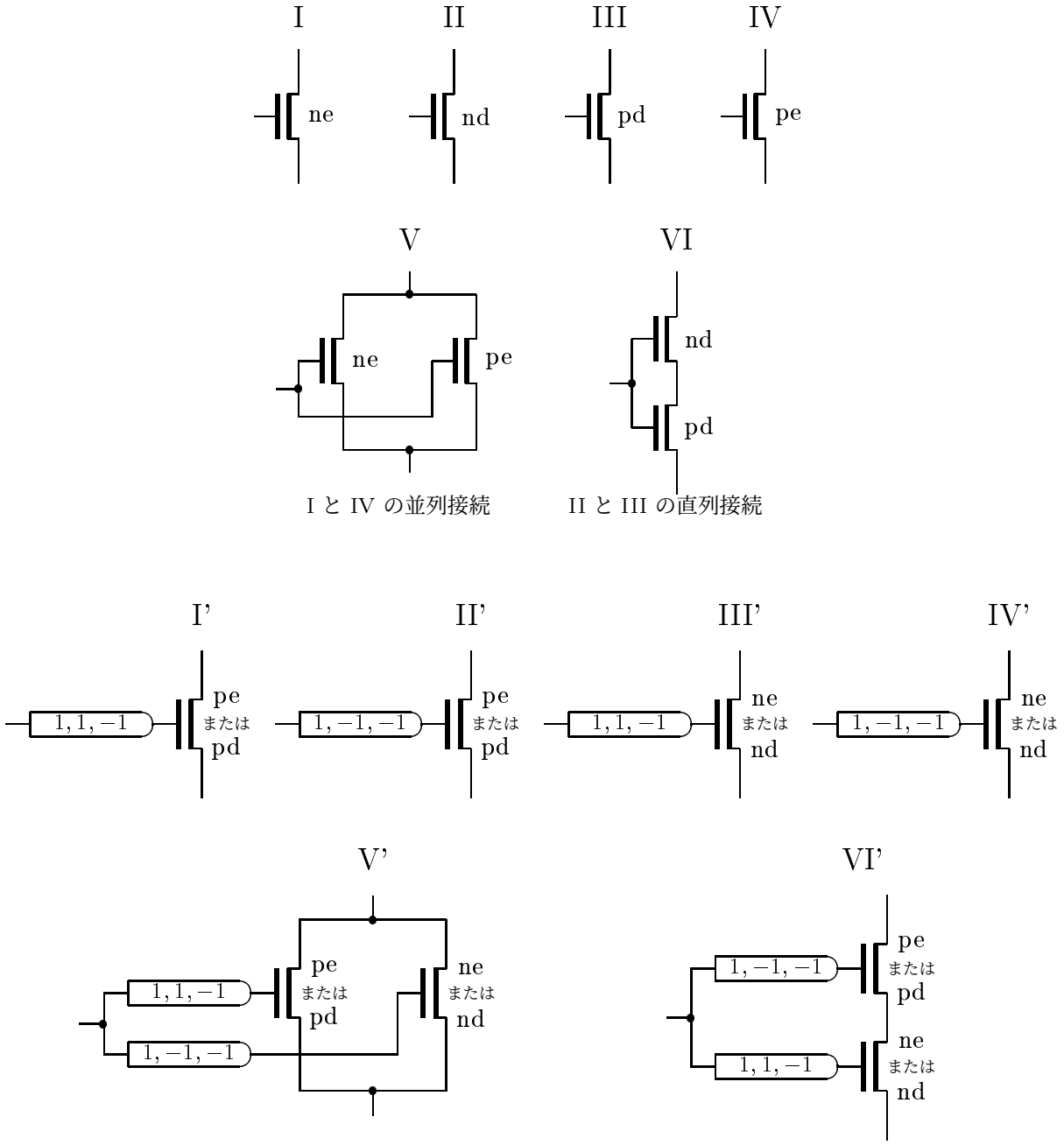


図 4.3: SW(0) で用いるスイッチ

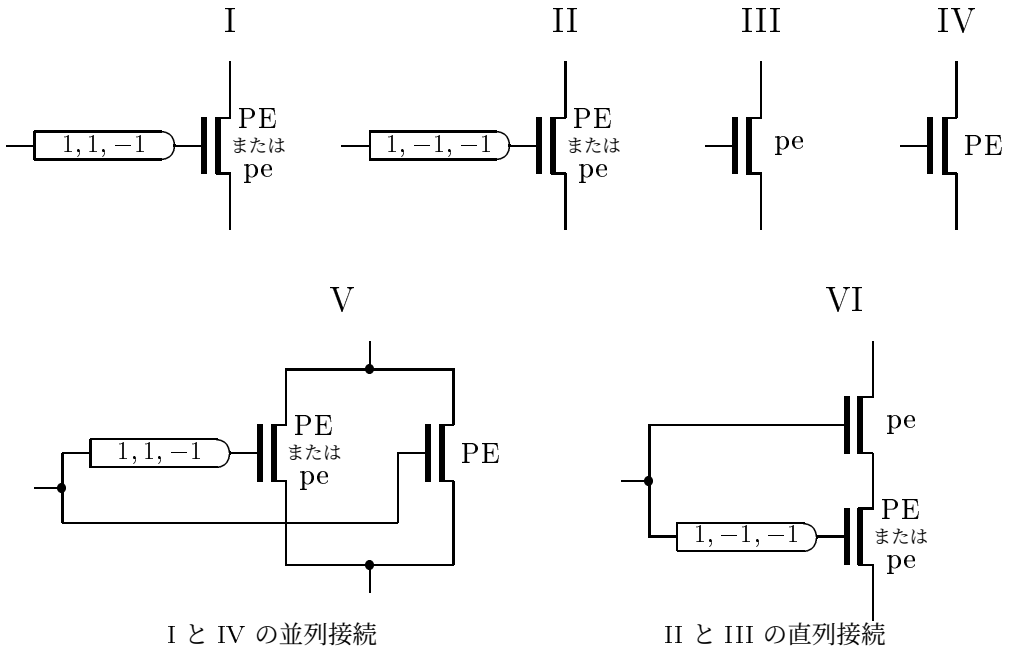


図 4.4: SW(1) で用いるスイッチ

SW(0) に関しては I から IV までのすべてを MOSFET で実現できるが，実際に各 MOSFET を使ってゲートを構成すると大きなリーク電流が流れる場合がある．このリーク電流を防ぐために，SW(−1) や SW(1) のときと同様に，(1, −1, −1) ゲートや (1, 1, −1) ゲートの出力を入力とするような方法をとらなければならないことがある．それらを I' や IV' などと表すことにする．I'〜VI' のスイッチ動作は I〜VI と同じである．

SW(−1)，SW(0)，SW(1) に必要なトランジスタとそれらの結合はそれぞれ図 4.2，4.3，4.4 のようになる．よって図 4.2〜図 4.4 のスイッチを直列/並列に接続することで，SW(−1)，SW(0)，SW(1) を構成でき，各 SW(n) の出力をワイヤードオアしたものを出力とすれば，必要な 3 値論理のためのゲートを実現できる．

ところで，各 SW(n) で用いられる MOSFET をまとめると，

SW(−1)	NE, ne
SW(0)	ne, nd, pe, pd
SW(1)	PE, pe

となる．ne と pe は 2 種類のスイッチで用いられている．[6] では，SW(-1) で用いられる ne と SW(0) で用いられる ne とを，SW(1) で用いられる pe と SW(0) で用いられる pe とを区別していない．しかしながら，MOSFET の動作は V_{gs} と閾値だけでなく，ソース電源電圧などの他の要素も関わっており [23, 24]，微細化加工がすすむにつれ，他の要素の影響が大きくなっている．実際に HSPICE レベル 49 でのシミュレーションでは

$$\text{SW}(-1)\text{での ne の閾値} > \text{SW}(0)\text{での ne の閾値}$$

とすると，全体的に 1 変数関数ゲートの性能が良くなった．pe に関しては，SW(1) で用いるものと SW(0) で用いるもので，閾値を変えてもほとんど効果がなかった．よって本論文では 7 種類の MOSFET を使って，ゲートをシミュレーションする．HSPICE のパラメーターは §4.5 で議論する．

4.3　3 値論理の 1 変数関数ゲート

3 値論理の 1 変数関数ゲートは 27 種類あるが，次の 5 つにクラス分けされる．

(i) 2 つのトランジスタで構成できる
このクラスのゲートは 2 値論理でのインバータに似ており，NMOSFET と PMOSFET 1 つずつを用いる．6 種類ある．

$$(0,-1,-1),\ (0,0,-1),\ (1,-1,-1),\ (1,0,0),\ (1,1,-1),\ (1,1,0)$$

なお，これらの関数を [6] ではリバース関数と呼んでいる．

(ii) 4 つのトランジスタで構成できる
このクラスのゲートはインバータ ($(1,0,-1)$ ゲート) だけである．

$$(1,0,-1)$$

(iii) (i) か (ii) を 2 段に組み合わせるもの
このクラスのゲートは 6 種類ある．

$$(-1,-1,0),\ (-1,-1,1),\ (-1,0,0),\ (-1,1,1),\ (0,0,1),\ (0,1,1)$$

各関数とも組合わせ方は 6 通りずつある．

(iv) 6 つのトランジスタで構成できるもの
このクラスのゲートは 10 種類あり，内部に $(1,1,-1)$ ゲートか $(1,-1,-1)$ ゲートを含んで

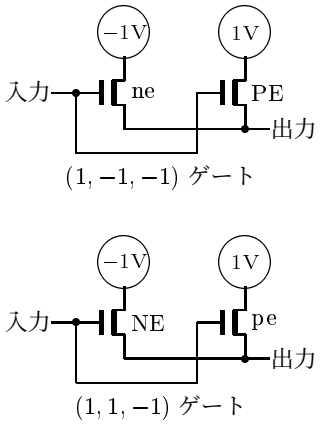


図 4.5: 2 つのトランジスタで構成できるゲートの例

いる．

$$(0,-1,0),\ (0,-1,1),\ (1,-1,0),\ (1,-1,1),\ (1,0,1),\ (0,1,0),\ (0,1,-1),\ (-1,1,0),$$
$$(-1,1,-1),\ (-1,0,-1)$$

(v) ゲートを必要としないもの

定数関数と，入力 = 出力となる関数がこのクラスになる．

$$(-1,-1,-1),\ (0,0,0),\ (1,1,1),\ (-1,0,1)$$

4.3.1 2 つのトランジスタで構成できるゲート

このクラスの関数には

$$(0,-1,-1),\ (0,0,-1),\ (1,-1,-1),\ (1,0,0),\ (1,1,-1),\ (1,1,0)$$

の 6 種類ある．このクラスのゲートの例として，$(1,-1,-1)$ のゲートを説明する．この関数は 0 を値にとらないので SW(0) は必要ない．SW(-1) と SW(1) の動作は次のようになる．

入力	SW(-1)	SW(1)
-1	off	on
0	on	off
1	on	off
動作	II	IV

よって，図 4.2 から SW(-1) には ne を，図 4.4 から SW(1) には PE を用いればよい．$(1,-1,-1)$ ゲートと後でよく出てくる $(1,1,-1)$ ゲートを図 4.5 に示す．

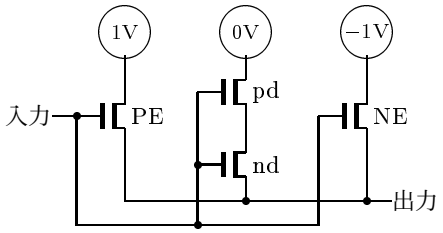


図 4.6: インバータ (NOT ゲート)

4.3.2 4 つのトランジスタで構成できるゲート

このクラスのゲートはインバータだけである．インバータの各スイッチ SW(n) のスイッチ動作は

入力	SW(-1)	SW(0)	SW(1)
-1	off	off	on
0	off	on	off
1	on	off	off
動作	I	VI	IV

となる．SW(-1) の動作は I なので図 4.2 から NE を用いることが分かる．SW(0) の動作は VI なので図 4.3 から nd と pd を直列にした回路であり，SW(1) の動作は IV なので図 4.4 から PE となる．よってインバータ (NOT ゲート) は図 4.6 となる．

4.3.3 トランジスタ 2 つのゲートかインバータを組み合わせるゲート

このクラスのゲートは

$$(-1,-1,0),\ (-1,-1,1),\ (-1,0,0),\ (-1,1,1),\ (0,0,1),\ (0,1,1)$$

の 6 種類があり，トランジスタ数は 4 か 6 になる．各関数とも組合わせ方は 6 種類ある (表 4.2)．

$(-1,-1,0)$ の 1 のところは $(1,1,0)$，$(0,0,-1)$ となっているが，これは前段に $(1,1,0)$ ゲート，後段に $(0,0,-1)$ ゲートをおくと $(-1,-1,0)$ ゲートが構成できることを意味している．組み合わせの選び方は，リーク電流やゲートの性能を考慮して決定すれば良い．

4.3.4 トランジスタを 6 個用いるゲート

このクラスのゲートは 10 種類ある．

$(0,-1,0)$, $(0,-1,1)$, $(1,-1,0)$, $(1,-1,1)$, $(1,0,1)$, $(0,1,0)$, $(0,1,-1)$, $(-1,1,0)$, $(-1,1,-1)$, $(-1,0,-1)$

これらの関数のゲートに共通することは，いくつかのトランジスタへの入力の前に $(1,-1,-1)$ ゲートか $(1,1,-1)$ ゲートを置かなければならないことである．このクラスの例として $(1,-1,1)$ ゲートと $(-1,1,-1)$ ゲートを説明する．$(1,-1,1)$ ゲートでは，SW(-1) と SW(1) は次のようになる．

入力	SW(-1)	SW(1)
-1	off	on
0	on	off
1	off	on
動作	VI	V

表 4.2: トランジスタ 2 つのゲートかインバータを組み合わせるゲートの組合わせ方

関数	$(-1,-1,0)$		$(-1,-1,1)$		$(-1,0,0)$	
	前段	後段	前段	後段	前段	後段
1	$(1,1,0)$,	$(0,0,-1)$	$(1,1,0)$,	$(1,1,-1)$	$(1,0,0)$,	$(0,0,-1)$
2	$(1,1,-1)$,	$(0,0,-1)$	$(1,1,-1)$,	$(1,1,-1)$	$(1,-1,-1)$,	$(0,0,-1)$
3	$(1,1,-1)$,	$(0,-1,-1)$	$(1,1,-1)$,	$(1,-1,-1)$	$(1,-1,-1)$,	$(0,-1,-1)$
4	$(0,0,-1)$,	$(0,-1,-1)$	$(0,0,-1)$,	$(1,-1,-1)$	$(0,-1,-1)$,	$(0,-1,-1)$
5	$(1,1,0)$,	$(1,0,-1)$	$(1,1,-1)$,	$(1,0,-1)$	$(1,0,0)$,	$(1,0,-1)$
6	$(1,0,-1)$,	$(0,-1,-1)$	$(1,0,-1)$,	$(1,-1,-1)$	$(1,0,-1)$,	$(0,0,-1)$
関数	$(-1,1,1)$		$(0,0,1)$		$(0,1,1)$	
	前段	後段	前段	後段	前段	後段
1	$(1,0,0)$,	$(1,1,-1)$	$(1,1,0)$,	$(1,1,0)$	$(1,0,0)$,	$(1,1,0)$
2	$(1,-1,-1)$,	$(1,1,-1)$	$(1,1,-1)$,	$(1,1,0)$	$(1,-1,-1)$,	$(1,1,0)$
3	$(1,-1,-1)$,	$(1,-1,-1)$	$(1,1,-1)$,	$(1,0,0)$	$(1,-1,-1)$,	$(1,0,0)$
4	$(0,-1,-1)$,	$(1,-1,-1)$	$(0,0,-1)$,	$(1,0,0)$	$(0,-1,-1)$,	$(1,0,0)$
5	$(1,-1,-1)$,	$(1,0,-1)$	$(0,0,-1)$,	$(1,0,-1)$	$(0,-1,-1)$,	$(1,0,-1)$
6	$(1,0,-1)$,	$(1,1,-1)$	$(1,0,-1)$,	$(1,0,0)$	$(1,0,-1)$,	$(1,1,0)$

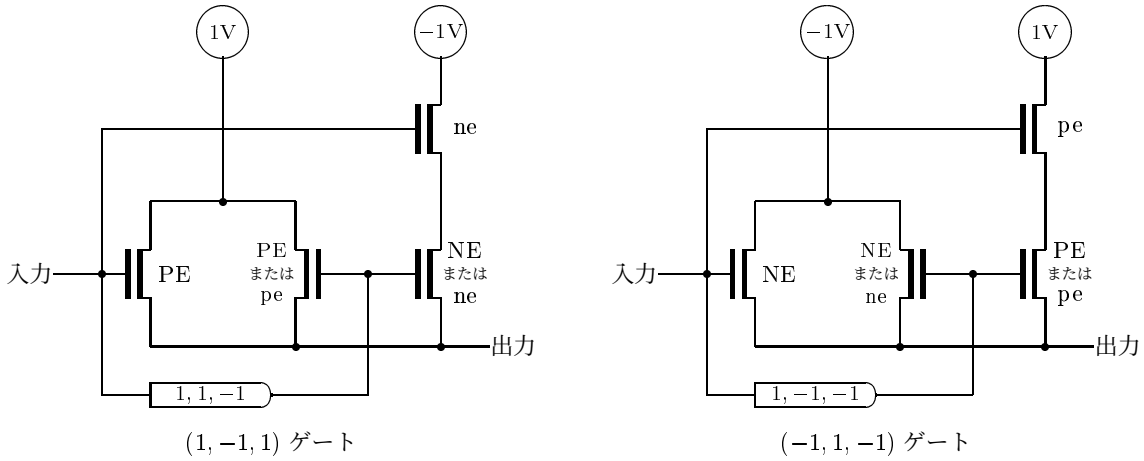


図 4.7: トランジスタを 6 個用いるゲート

このゲートは SW(−1) には図 4.2 の VI を用い，SW(1) には図 4.4 の V を用いる．

$(-1, 1, -1)$ ゲートでは，SW(−1) と SW(1) は次のようになる．

入力	SW(−1)	SW(1)
−1	on	off
0	off	on
1	on	off
動作	V	VI

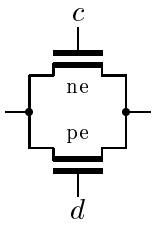
$(-1, 1, -1)$ ゲートでは SW(−1) には図 4.2 の V を用い，SW(1) には図 4.4 の VI を用いる．$(1, -1, 1)$ ゲートと $(-1, 1, -1)$ ゲートは図 4.7 のようになる．

4.3.5 ゲートを必要としない関数

定数関数 $(-1, -1, -1)$ のゲートはすべての入力に対して -1 を出力するので，-1 V の電源電圧を用いれば良い．同じく，$(0, 0, 0)$ には 0 V の電源電圧を，$(1, 1, 1)$ には 1 V の電源電圧を用いる．よって，これらの関数にはゲートは必要ない．$(-1, 0, 1)$ は 入力=出力 なので，特にゲートを必要としない．

4.3.6　T ゲートと縮退 T ゲート

本論文は T ゲートをトランスファーゲートを用いて構成することを提案する．トランスファーゲートは ne と pe を組み合わせて構成する．



このゲートの動作は次のようになる．なお，HSPICE のシミュレーション結果より，ne にはソース電圧が 0V のときに使用する方 (閾値が通常の ne より低いもの) を用いることにする．

c	d	動作
1	-1	導通状態
-1	1	カットオフ
その他		不定

トランスファーゲートの制御信号は 1 変数関数ゲートで生成できるので，3 つのトランスファーゲートと 6 つの 1 変数関数ゲートを使って T ゲートを構成できる (図 4.8)．すると T ゲートに必要なトランジスタ数は 30 となり，これは決して少ない数ではない．そのためトランジスタ数を削減する工夫が必要である．図 4.9 のようにすると，トランジスタ数を 16 まで削減することができる．

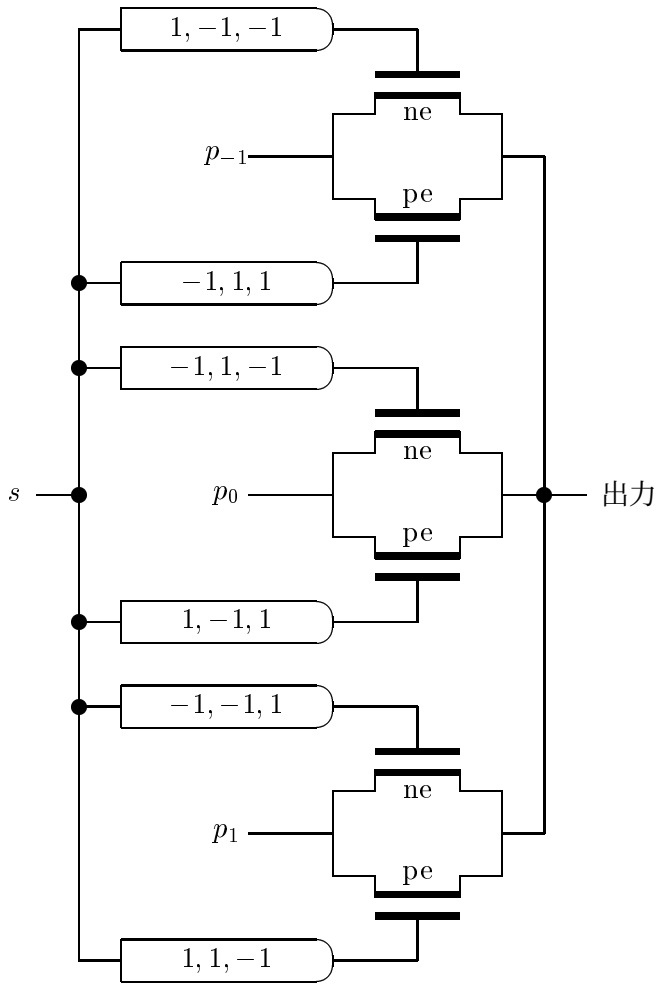
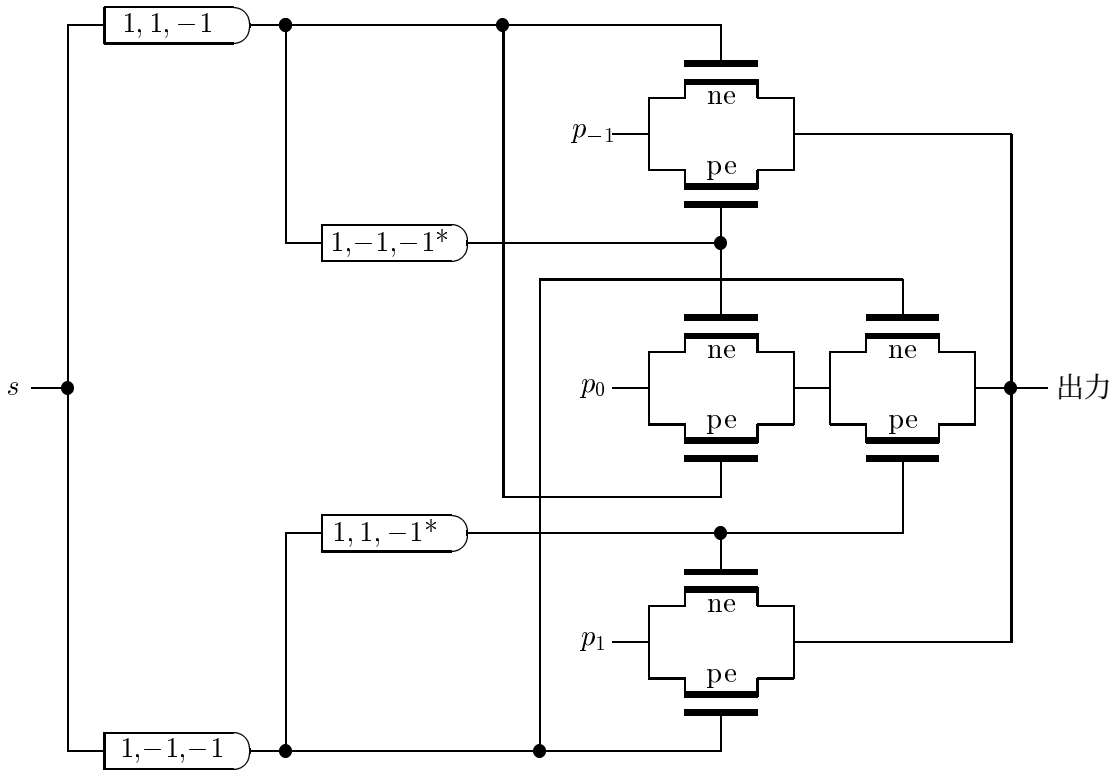


図 4.8: トランスファーゲートを用いる T ゲート



*の付いている $(1,-1,-1)$ ゲートは $(1,1,-1)$ ゲートに、$(1,1,-1)$ ゲートは $(1,-1,-1)$ ゲートに置き換えても、機能的には変わらないが、HSPICE のシミュレーション結果によりこれらのゲートを選んだ。

図 4.9: トランスファーゲートを用いる改良された T ゲート

縮退 T 演算子は 6 種類あるが，$T_{\overline{-1}}$ ゲートは T_{-10} ゲートと同じ構成にすることができる．同様に $T_{\overline{1}}$ ゲートは T_{01} ゲートと同じ構成にすることができる．$T_{\overline{0}}$ ゲートとしては T_{01} ゲートと T_{01} ゲートの両方を用いることができる．よって縮退 T ゲートは実質的には 3 種類である．これらの縮退 T ゲートの一例は図 4.10〜4.12 のようになり，トランジスタ数は

T_{-10}	T_{-11}	T_{01}	$T_{\overline{-1}}$	$T_{\overline{0}}$	$T_{\overline{1}}$
6	12	6	6	6	6

となる．

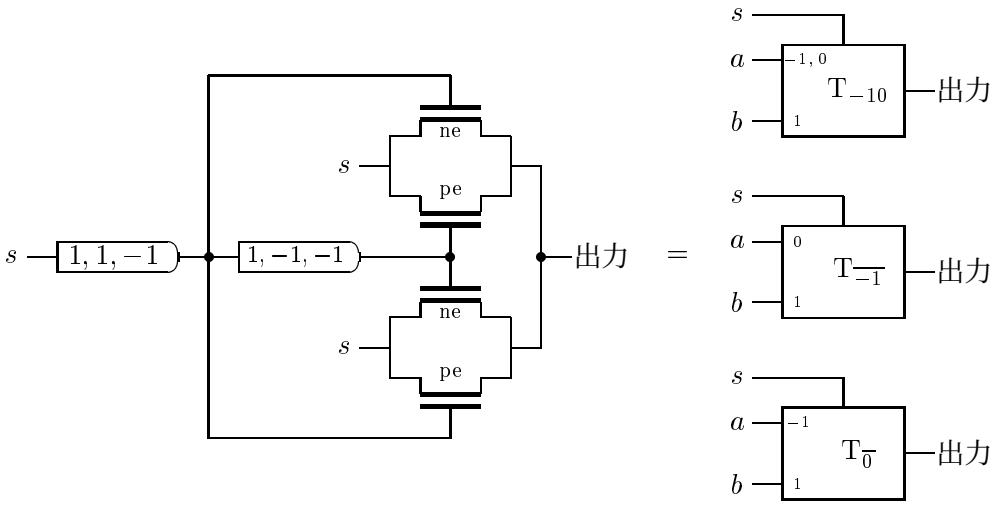


図 4.10: T_{-10} ゲート=$T_{\overline{-1}}$ ゲート=$T_{\overline{0}}$ ゲートの一例

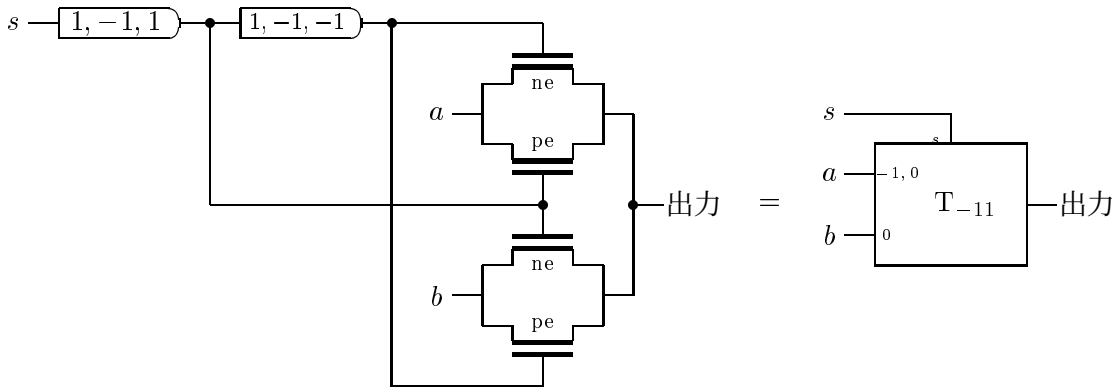


図 4.11: T_{-11} ゲートの一例

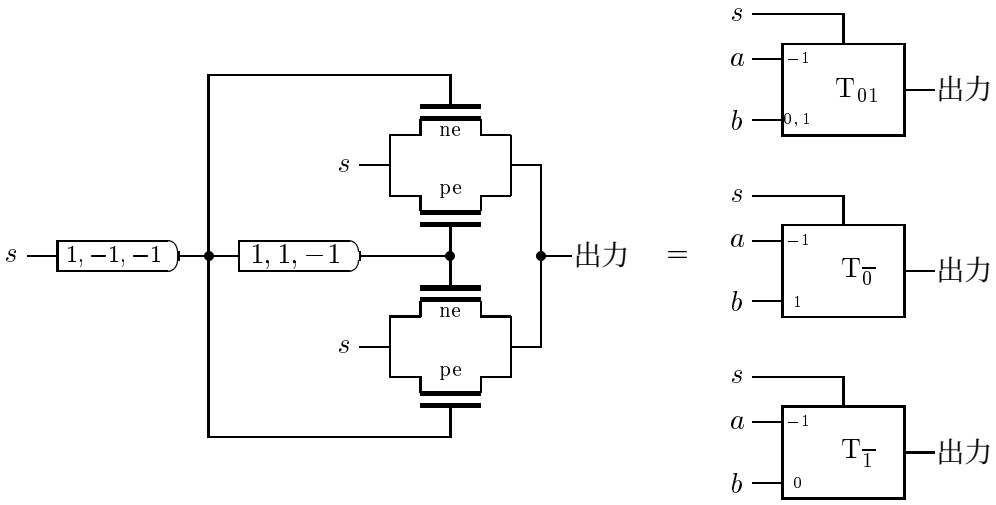


図 4.12: T_{01} ゲート=$T_{\overline{0}}$ ゲート=$T_{\overline{1}}$ ゲートの一例

4.4 TG 法

以上から 3 値論理の 1 変数関数ゲートと T ゲートを構成できることが分かった．§4.1.3 で示したように，1 変数関数ゲートと T ゲートから，任意の 2 変数関数ゲートを構成できる．1 変数関数は 27 種類しかなく，そのうち 4 種類はゲートを必要とせず，6 種類は残りの関数のゲートを組み合わせるので，実質的には 17 種類の 1 変数関数ゲートと T ゲートにより，任意の 2 変数関数ゲートを構成できる．3 値の場合，2 変数関数の種類は $3^{3^2} = 19683$ 種類もあることを考えると，18 個のモジュールで 2 変数関数ゲートを構成できるのは，効率的である．Olson 法による 1 変数関数ゲートと，トランスファーゲートによる T ゲートをモジュールとして，任意の 3 値論理関数ゲートを組織的に構成する本方法を TG 法 (トランスファーゲート法) と呼ぶことにする．また，縮退 T ゲートも用意すれば，ゲートに必要なトランジスタ数を削減できる．

4.5 HSPICE のパラメータ

本論文ではシミュレーションに HSPICE レベル 49 を用いた．レベル 49 では $0.1\mu m$ プロセスのゲートを正確にシミュレーションできる．HSPICE はプロセッサの微細化が進むに従いレベルを上げてきており，それに伴い，シミュレーションで指定するパラメータの種類も増えてきた．$0.35\mu m$ プロセスのような古いプロセスに対しては，商品用に用いられた物質特性パラメータが公開されているが，$0.1\mu m$ や $0.07\mu m$ プロセスのような新しいプロセスに対しては，公開されていない．そのため本論文では，HSPICE の開発者である Electrical Engineering and Computer Science,

University of California, Berkeley が公開している「Berkeley Predictive Technology Model」[25] の $0.1\mu m$ プロセスを用いることにした[1]．このモデルは実際の商品のために用いられるパラメータほどには実際的ではないかもしれないが，かなり高く信用できる．なお，形状パラメータに関しては

```
m0 x0 x1 x2 x2 pch
+ l=0.100000u
+ w=0.600000u
+ ad=0.180000e-12
+ as=0.180000e-12
+ pd=1.800000e-6
+ ps=1.800000e-6
+ nrd=0.090000e-12
+ nrs=0.090000e-12

m1 y0 y1 y2 y2 nch
+ l=0.100000u
+ w=0.400000u
+ ad=0.120000e-12
+ as=0.120000e-12
+ pd=1.400000e-6
+ ps=1.400000e-6
+ nrd=0.060000e-12
+ nrs=0.060000e-12
```

を用いた．

4.5.1 閾値

§4.2 で説明したように 3 値論理ゲートには閾値の調整が必要である．ここでは，閾値に関係するパラメータについて説明する．HSPICE では閾値 V_{th} の関係式

$$V_{th} = V_{th\,0} + K1(\sqrt{\phi_S - V_{bs}} - \sqrt{\phi_S}) - K2\,V_{bs} + K1\left(\sqrt{1 + \frac{N_{lx}}{L_{eff}}\sqrt{\frac{\phi_S}{\phi_S - V_{bs}}}} - 1\right)\sqrt{\phi_S}$$

[1] 以前は [25] から本論文で使用した物質特性パラメータについて書かれているページへのリンクが張られていたが，最近パラメータが更新されており，$0.1\mu m$ プロセスのパラメータが見られなくなった．そのため本論文で用いた $0.1\mu m$ プロセスのパラメータを付録として掲載する．

$$+(K3+K3B\cdot V_{bs})\left(\frac{T_{ox}}{W_{eff}+W_0}\right)\phi_S-\Delta V_{th}$$

を用いている．各パラメータの意味は [24] を参照．ここで V_{th0} は V_{bs}(バルク·ソース間電圧) $=0$ 及び小電圧 V_{ds} における長チャネルデバイスの閾値電圧であり，$V_{bs}=0$ とすると，

$$V_{th0}=V_{fb}+\Phi_s+K_1\sqrt{\phi_s} \tag{4.1}$$

という関係がある．なお，常に $V_{bs}=0$ となるように MOSFET を構成することはできる．ϕ_s は N_{th}(界面近傍のピークドレイン濃度) の関数であるから，V_{th0} も N_{th} の関数である．一般に，イオン注入時間を調節することで N_{ch} を変えることにより，MOSFET の閾値の調整を行う．

よって，本論文では N_{ch} と (4.1) の計算によって得られる V_{th0} を指定することで，シミュレーションに用いる閾値を定めた．残りのパラメータは Berkeley Predictive Technology Model の $0.1\mu m$ プロセスをそのまま用いた．また論理値 -1 には -0.3V，論理値 0 には 0V 論理値 1 には 0.3V を対応させることとした．このように選択した理由は，これより大きな電源電圧では，N_{ch} をより大きな値にしなければならないが，そうすると HSPICE が正しく動作しなかったからである．

4.6 単独トランジスタのパラメータとその特性

HSPICE でのシミュレーションの結果から，N_{ch} と V_{th0} の値を次のようにした．

NMOSFET			PMOSFET		
名前	$N_{ch}(10^{17}\mathrm{cm}^{-3})$	V_{th0}(V)	名前	$N_{ch}(10^{17}\mathrm{cm}^{-3})$	$V_{th0}(V)$
NE	30	0.3308	PE	14.6	-0.3239
ne(-1)	2	0.1620	pe	2.38	-0.2110
ne(0)	1.19	0.1293			
nd	0.197	0.0089	pd	0.595	-0.1239

原則的に，ne(-1) は電源電圧が -0.3V のときに用い，ne(0) は電源電圧が 0V のときに用いる ne である．またトランスファーゲートには ne(0) を用いる．このようにパラメータを選ぶと各トランジスタのスイッチ特性は図 4.13 と図 4.14 のようになる．

4.7 4 章のまとめ

4 章では MOSFET の閾値の調整と複数電源の使用以外は，従来の CMOS デバイスと同じ行程で 3 値論理ゲートを製造できる Olson 法を説明し，Olson 法の改良である TG 法を提案した．MOSFET の閾値の調整と複数電源の使用は，消費電力の削減のために，現在ではよく用いられる技術となっている．TG 法は，Olson 法によって構成した 1 変数関数ゲートと T ゲートを組み合わ

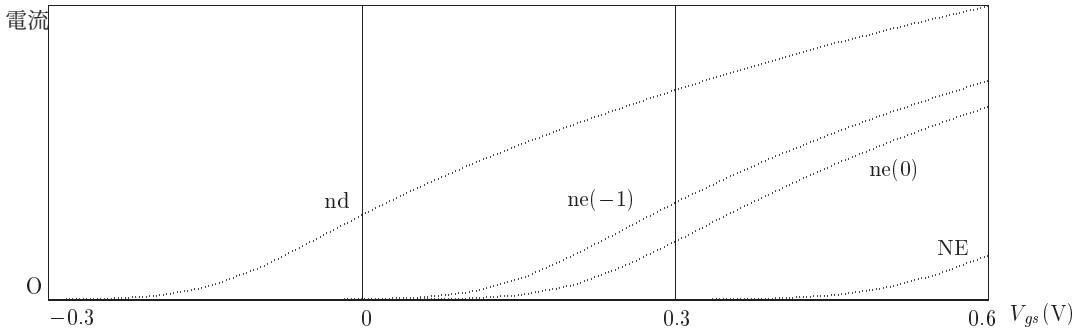


図 4.13: NMOSFET のスイッチ特性

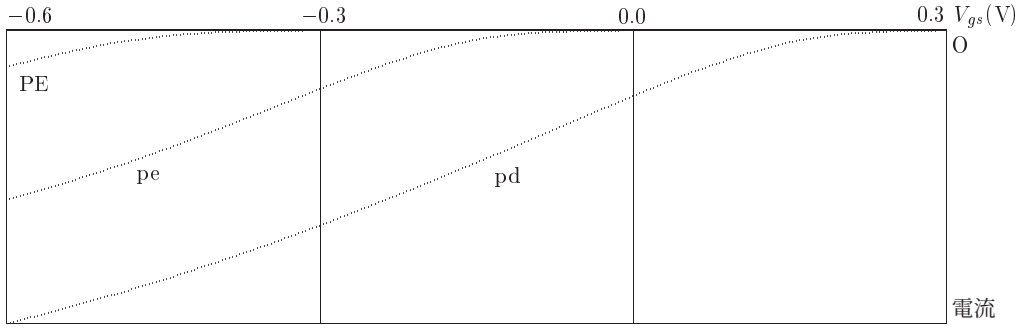


図 4.14: PMOSFET のスイッチ特性

せることで，2 変数関数ゲートを構成する手法である．TG 法では，1 変数関数ゲートと T ゲートをモジュールとして用意しておけば，任意の 2 変数の 3 値論理ゲートを構成できる．Olson 法や TG 法による 3 値論理ゲートは，定常状態では電流が流れないという CMOS デバイスの特性も受け継いでいる．つまり，この章で扱ったゲート構成法では，従来とほとんど同じ技術により製造でき，電力の消費の少ない 3 値論理ゲートを構成できる．また，TG 法での 2 変数ゲートの遅延も

1 変数関数ゲートの遅延 + トランスファーゲートの遅延

となる．Olson の方法では関数によってはトランジスタの直列接続の数が多くなるため，複雑な関数では TG 法によるゲートの方が性能が良くなる (表 5.3 参照).

また，4 章では閾値に関する 3 値論理ゲートのための HSPICE のパラメータを決定した．これらの値を用いて，5 章で設計するハードウェアの動作を HSPICE でシミュレーションする．

第 5 章

標数 3 の体での XTR のためのハードウェア

5.1 標数 3 の体での XTR のためのハードウェアの構成と演算器

一般に ElGamal 暗号のためのハードウェアは，図 5.1 のように，演算を行うための演算器，ハードウェアを制御するための命令が書かれている命令 ROM，コード生成のための命令シーケンサ，途中結果を記憶するためのレジスタから構成される．復号化では，暗号文 (C_1, C_2) に対して，C_1 の部分だけ処理して $d(C_1)$ を得，それから $d(C_1) \oplus C_2$ か $C_2/d(C_1)$ を行うことで復号文が得られ

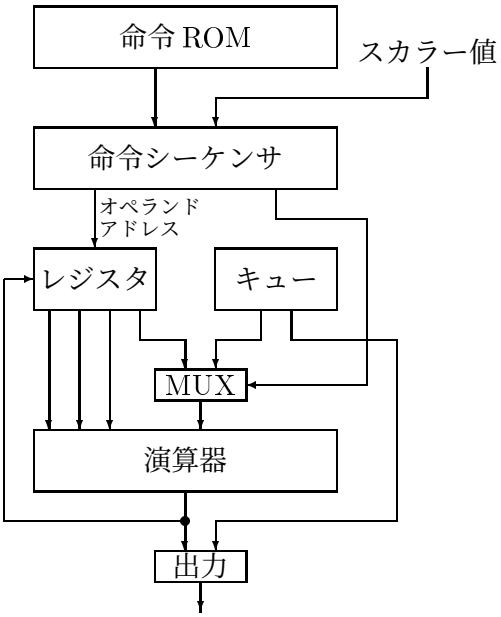


図 5.1: 標数 3 の体の XTR を含む ElGamal 暗号ハードウェアの概要

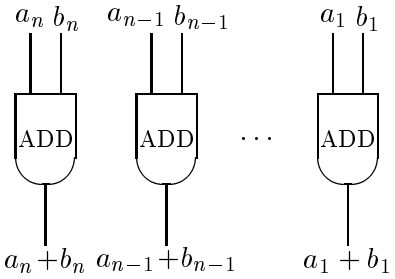


図 5.2: $\mathbb{F}_{3^n}$ の加算器

る．それで，暗号文の半分の C_2 を蓄積するためのキューを設けている．図 5.1 のスカラー値とは，暗号化で使う乱数値か，復号化のときに使う秘密鍵の値のことである．

3 章で論じたように，標数 3 の体での XTR では，演算は $\mathbb{F}_{3^n}$, (n は奇数) での加減算と乗算，及び 3 乗，$\sqrt{3^{n+1}}$ 乗だけで行うことができる．$\mathbb{F}_{3^n}$ が $\mathbb{F}_3$ 上タイプ II の ONB(§2.3.3) を持つ場合だけを考える．すると，$\mathbb{F}_{3^n}$ の元は $a = a_n(X^n + X^{-n}) + a_{n-1}(X^{n-1} + X^{-n+1}) + \cdots + a_1(X + X^{-1})$ という多項式の形をしている．ここで各 a_i は $\mathbb{F}_3$ の元である．a の元は計算機では $\mathbb{F}_3$ の n 個の元の組 $(a_n, a_{n-1}, \cdots, a_1)$ として扱われる．

5.1.1 標数 3 の体の加減算器

$\mathbb{F}_{3^n}$ の 2 元 $a = (a_n, a_{n-1}, \cdots, a_1)$ と $b = (b_n, b_{n-1}, \cdots, b_1)$ との和は $(a_n + b_n, a_{n-1} + b_{n-1}, \cdots, a_1 + b_1)$ となるので，n 個の ADD ゲートにより一度に $a + b$ を計算できる (図 5.2)．ところで加算器の入力と出力の値を反転できるようにすると，$\pm a \pm b$ を計算できるようになる．4 つの値 $\pm a \pm b$ から 1 つを選ぶために，2 桁の選択信号 s_0, s_1 を用いることとし，

s_0	s_1	結果
1	1	$a + b$
1	-1	$-a - b$
-1	1	$a - b$
-1	-1	$-a + b$

のように選択されるとすると，1 桁の加減算器は図 5.3 のようになる．この加減算器を n 個並列に並べれば，$\mathbb{F}_{3^n}$ の加減算 $\pm a \pm b$ を一度に計算できる．

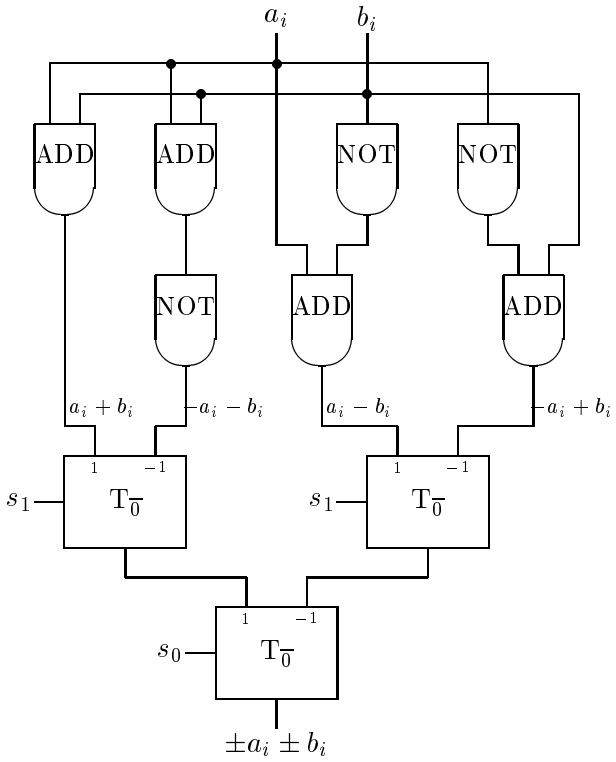


図 5.3: 標数 3 の体の 1 桁の加減算器

5.1.2 標数 3 の体の乗算器

$\mathbb{F}_{3^n}$ の元 $a = (a_n, a_{n-1}, \cdots, a_1)$, $b = (b_n, b_{n-1}, \cdots, b_1)$ に対して，積 ab は最初に $1 \leq i, j \leq n$ に対して部分積 $a_i b_j$ を生成し，それらから各基底に対していくつかを選択し，それらを足し合わせる．$\mathbb{F}_3$ の基底にタイプ II の ONB を用いているので，$a = (a_n, a_{n-1}, \cdots, a_1),\ b = (b_n, b_{n-1}, \cdots, b_1)$ に対して

$$a \cdot b = c = (c_n, c_{n-1}, \cdots, c_1)$$

とすると

$$\begin{aligned} c_k &= \sum_{i,j} a_i \cdot b_j - 2(a_1 b_1 + a_2 b_2 + \cdots + a_n b_n) \\ &= \sum_{i,j} a_i \cdot b_j + (a_1 b_1 + a_2 b_2 + \cdots + a_n b_n) \quad (\text{標数 } 3 \text{ より}) \\ &= \sum_{i,j} a_i \cdot b_j + A \end{aligned} \tag{5.1}$$

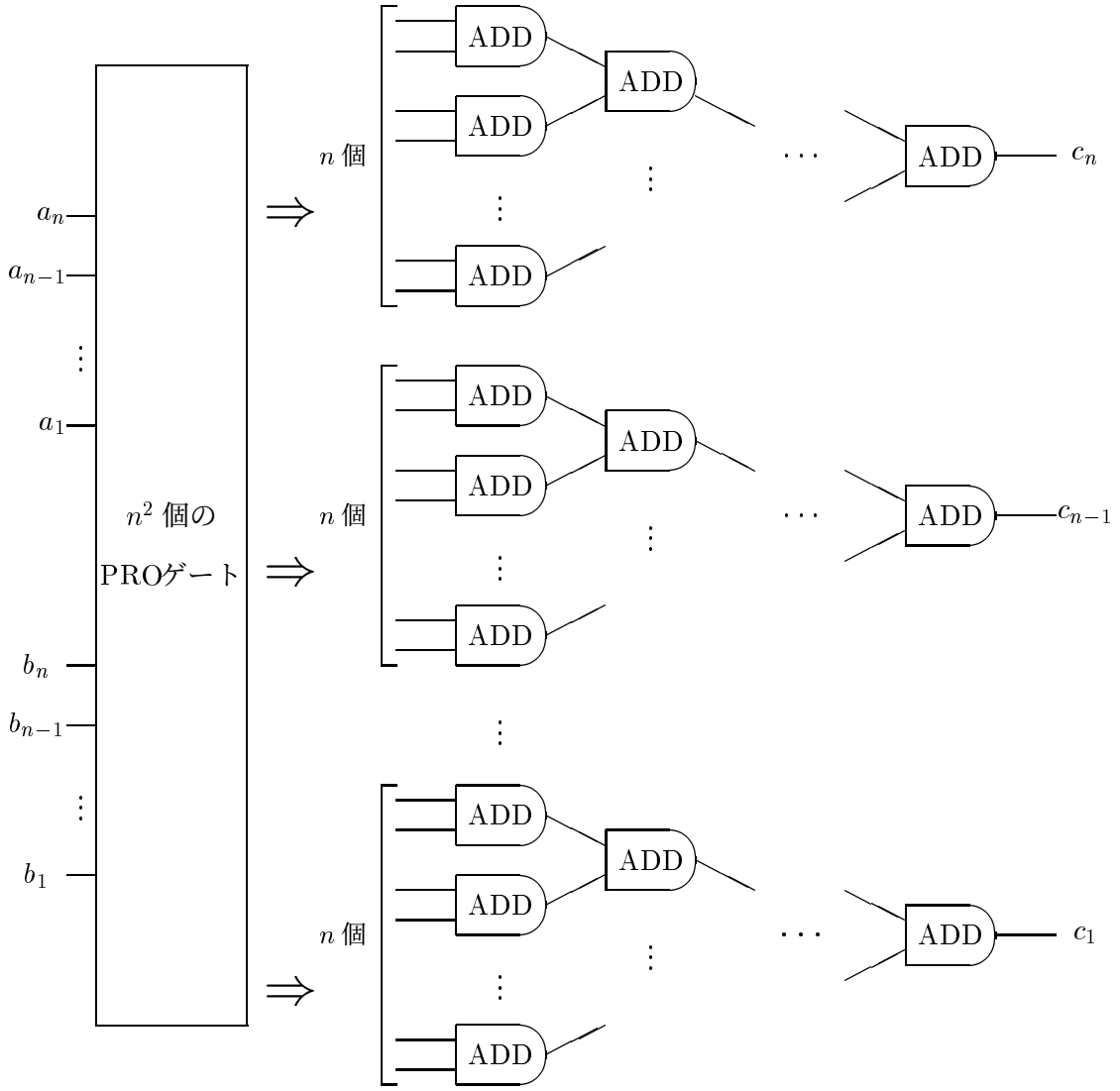


図 5.4: $\mathbb{F}_{3^n}$ の乗算器

となる．ここで $A = a_1b_1 + a_2b_2 + \cdots + a_nb_n$ であり，$\sum$ は次の 4 つのうちの 1 つをみたすような (i, j) に対して加算を行う．

$$\begin{aligned} i + j &= k \text{ または } 2n + 1 + k \\ 2n + 1 + i - j &= k \text{ または } 2n + 1 + k \\ 2n + 1 - i + j &= k \text{ または } 2n + 1 + k \\ 4n + 2 - i - j &= k \text{ または } 2n + 1 + k \end{aligned}$$

各 k に対して (5.1) の項の個数は $2n$ となり，各 $a_i,\ b_j$ はそれぞれ $2n$ 個ずつ使用される．

部分積 a_ib_j の計算には，n^2 個の PRO ゲートがあれば良い．部分積を足し合わせる部分は，ADD ゲートを葉の数が n の逆向きの二分木のように並べればよい (図 5.4)．n が 2 のべき $(n = 2^{n'})$ なら

ば，1つの基底に対する乗算結果を求めるのにADDゲートは$1+2+2^2+\cdots+2^{n'}=2^{n'+1}-1=2n-1$個必要であるから，加算器全体で必要となるADDゲートは$2n^2-n$個必要になる．3値論理ゲートを用いることで，部分積を求める部分の配線を簡単にでき，部分積を足す部分は，配線の交差がないように配置できる．

5.1.3 シフタ

標数3のXTRでは，$\mathbb{F}_{3^n}$の3乗計算と$\sqrt{3^{n+1}}$計算が必要であるが，これらの計算はシフトで行うことができる．例えば$n=111$のときは，表5.1のようにシフトを行えば良い．

表 5.1: $n=111$のときの3乗と$\sqrt{3^{n+1}}$乗

i	3乗	$\sqrt{3^{n+1}}$乗	i	3乗	$\sqrt{3^{n+1}}$乗	i	3乗	$\sqrt{3^{n+1}}$乗	i	3乗	$\sqrt{3^{n+1}}$乗	i	3乗	$\sqrt{3^{n+1}}$乗
1	3	79	26	78	47	51	70	15	76	5	17	101	80	49
2	6	65	27	81	97	52	67	94	77	8	62	102	83	30
3	9	14	28	84	18	53	64	50	78	11	82	103	86	109
4	12	93	29	87	61	54	61	29	79	14	3	104	89	35
5	15	51	30	90	83	55	58	108	80	17	76	105	92	44
6	18	28	31	93	4	56	55	36	81	20	68	106	95	100
7	21	107	32	96	75	57	52	43	82	23	11	107	98	21
8	24	37	33	99	69	58	49	101	83	26	90	108	101	58
9	27	42	34	102	10	59	46	22	84	29	54	109	104	86
10	30	102	35	105	89	60	43	57	85	32	25	110	107	7
11	33	23	36	108	55	61	40	87	86	35	104	111	110	72
12	36	56	37	111	24	62	37	8	87	38	40			
13	39	88	38	109	103	63	34	71	88	41	39			
14	42	9	39	106	41	64	31	73	89	44	105			
15	45	70	40	103	38	65	28	6	90	47	26			
16	48	74	41	100	106	66	25	85	91	50	53			
17	51	5	42	97	27	67	22	59	92	53	91			
18	54	84	43	94	52	68	19	20	93	56	12			
19	57	60	44	91	92	69	16	99	94	59	67			
20	60	19	45	88	13	70	13	45	95	62	77			
21	63	98	46	85	66	71	10	34	96	65	2			
22	66	46	47	82	78	72	7	110	97	68	81			
23	69	33	48	79	1	73	4	31	98	71	63			
24	72	111	49	76	80	74	1	48	99	74	16			
25	75	32	50	73	64	75	2	96	100	77	95			

$a\in\mathbb{F}_{3^{111}}$に対して，$b=a^3$，$c=a^{\sqrt{3^{n+1}}}$とする．$a=(a_{111},a_{110},\cdots,a_2,a_1)$，$b=(b_{111},b_{110},\cdots,b_2,b_1)$，$c=(c_{111},c_{110},\cdots,c_2,c_1)$であるとし，上の表で

i	3乗	$\sqrt{3^{n+1}}$乗
i'	j	k

となっているとすると，$b_j=a_{i'},c_k=a_{i'}$となる．

5.1.4 ラッチ

暗号ハードウェアに必要なレジスタやパイプライン・レジスタのためにラッチが必要になる．ここでは，3 値論理のラッチについて提案する．2 値論理のラッチは，図 5.5 のようにインバータとトランスファーゲートを用いて構成されるが，3 値論理でも同様にインバータ (NOT ゲート) とトランスファーゲートを用いて構成できる (図 (5.6))．

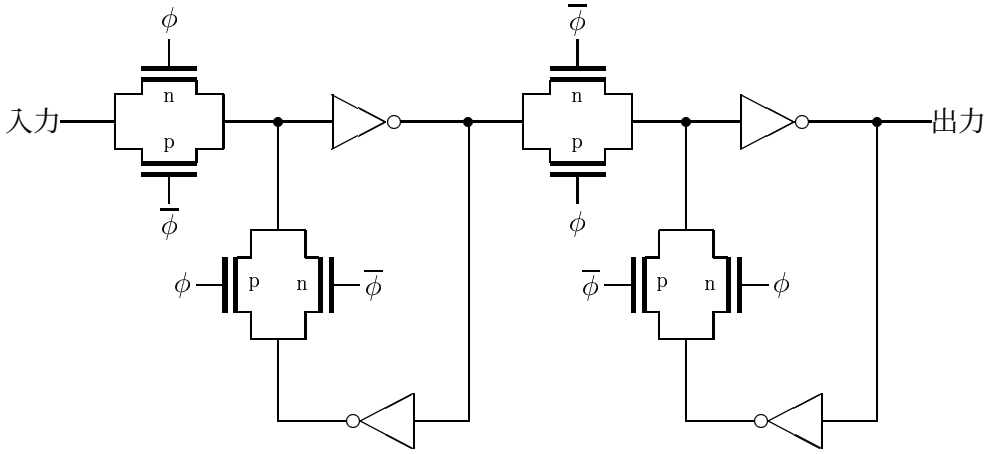


図 5.5: 2 値論理のラッチ

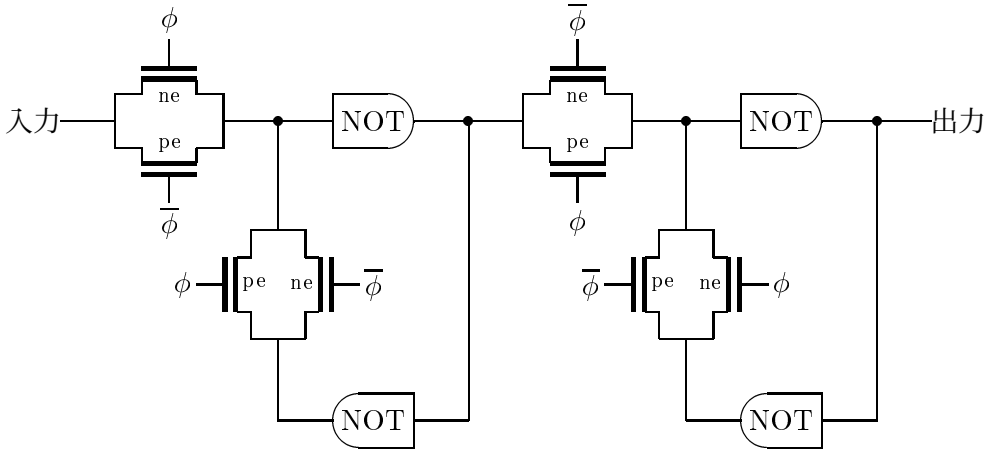


図 5.6: 3 値論理のラッチ

5.2 標数 3 の体での XTR の計算の手順

標数 3 の体での XTR では，暗号化も復号化も $d \in \mathbb{F}_{3^n}$ と乱数値または秘密鍵 m から $d_m \in \mathbb{F}_{3^n}$ を求めることを行う．d_m の計算は定理 3.6 の手順で計算できる．ここで d から d_m を計算するための詳しい手順を示す．用いる演算は加減算，乗算，3 乗計算，$\sqrt{3^{n+1}}$ 乗計算だけとし，用いる値は d，m，1，0 だけとする．また，各変数は $\mathbb{F}_{3^n}$ の元を表しているとする．アルゴリズム 3.2 の各値は $\mathbb{F}_{3^{2n}}$ の元であり，$\mathbb{F}_{3^n}$ の元 2 つで表現される．§3.4 と同様に $\mathbb{F}_{3^{2n}}/\mathbb{F}_{3^n}$ の基底には $\{1, \sqrt{-1}\}$ を用いることとし，この節では $\mathbb{F}_{3^{2n}}$ の元 $a + b\sqrt{-1}$ を (a, b) と書くことにする．(C_{00}, C_{01}), (C_{10}, C_{11}), (C_{20}, C_{21}), (C_{30}, C_{31}), (T_{10}, T_{11}), (T_{20}, T_{21}), (T_{30}, T_{31}) はそれぞれアルゴリズム 3.2 の $C[0]$, $C[1]$, $C[2]$, $C[3]$, $T[1]$, $T[2]$, $T[3]$ に対応しているとする．また，$\sqrt{-1} = (0, 1)$ である．

d と m から d_m を計算するには定理 3.6 を用いる．また定理 3.6 は 3 つのステップから成っている．定理 3.6 の各ステップについて見ていく．なお標数 3 の体では $2x = -x$ となることを使っている．

定理 3.9 のステップ (i) では d から c の計算を行う．

	演算			注釈
1.	x	$\leftarrow$	$d \times d$	
2.	y	$\leftarrow$	$d^{\sqrt{3^{n+1}}}$	シフタを使う
3.	x	$\leftarrow$	$-x + y$	
4.	y	$\leftarrow$	$1 + 0$	4, 5, 6 はバイナリ法による $(q+1)/4$ 乗計算を行う
5.	y	$\leftarrow$	$x \times y$	
6.	y	$\leftarrow$	$x \times k$	$(3^n + 1)/4$ のビット値によっては 6 は行わない
	5, 6 を繰り返す			$y = (2d^2 + d^{\sqrt{3^{n+1}}})^{(3^n+1)/4}$ となる
7.	C_{30}	$\leftarrow$	$-d + 0$	
8.	C_{31}	$\leftarrow$	$y + 0$	$(C_{30}, C_{31}) = c = C[3]$ となる

定理 3.6 のステップ (ii) ではアルゴリズム 3.2 を使って c と m から c_m を計算する．アルゴリズム 3.2 も 4 つのステップに分かれているので，アルゴリズムの各ステップ毎に見ていく．

アルゴリズム 3.2 のステップ 1 は初期値設定である．

	演算			注釈
1.	C_{00}	$\leftarrow$	C_{30}^3	シフタを使う
2.	C_{01}	$\leftarrow$	C_{31}^3	シフタを使う, $C[0] = (C_{10}, C_{11})$
3.	x	$\leftarrow$	$C_{00} - C_{30}$	
4.	x	$\leftarrow$	$x \times C_{30}$	
5.	y	$\leftarrow$	$C_{01} + C_{31}$	
6.	y	$\leftarrow$	$y \times C_{31}$	
7.	C_{10}	$\leftarrow$	$x - y$	
8.	x	$\leftarrow$	$C_{10} - C_{31}$	
9.	x	$\leftarrow$	$x \times C_{30}$	
10.	y	$\leftarrow$	$C_{00} + C_{30}$	
11.	y	$\leftarrow$	$y \times C_{31}$	
12.	C_{11}	$\leftarrow$	$x + y$	$(C_{10}, C_{11}) = C[1]$
13.	x	$\leftarrow$	$C_{00} - C_{01}$	
14.	y	$\leftarrow$	$C_{00} + C_{01}$	
15.	x	$\leftarrow$	$x \times y$	
16.	C_{20}	$\leftarrow$	$x + C_{00}$	
17.	x	$\leftarrow$	$C_{00} \times C_{01}$	
18.	C_{21}	$\leftarrow$	$x - C_{01}$	$(C_{20}, C_{21}) = C[2]$

アルゴリズム 3.2 のステップ 2 では m からのビット列 m_j を生成する．ステップ 3 はループであるが，ループ 1 回あたりは次のようになる．なおループの回数は $|m| - 2$ 回である．また，j に対して，$k = m_j$, $l = m_j + 1$ であるとする．計算には命題 3.3 を使っている．

	演算			注釈
1.	x	$\leftarrow$	$C_{k0} - C_{k1}$	
2.	y	$\leftarrow$	$C_{k0} + C_{k1}$	
3.	x	$\leftarrow$	$x \times y$	
4.	T_{10}	$\leftarrow$	$x + C_{k0}$	
5.	x	$\leftarrow$	$C_{k0} \times C_{k1}$	
6.	T_{11}	$\leftarrow$	$c - C_{k1}$	$(T_{10}, T_{11}) = T[1]$
7.	x	$\leftarrow$	$C_{l0} - C_{l1}$	
8.	y	$\leftarrow$	$C_{l0} + C_{l1}$	
9.	x	$\leftarrow$	$x \times y$	
10.	T_{20}	$\leftarrow$	$x + C_{l0}$	
11.	x	$\leftarrow$	$C_{l0} \times C_{l1}$	
12.	T_{21}	$\leftarrow$	$x - C_{l1}$	$(T_{20}, T_{21}) = T[2]$

ステップ 13〜24 は k の値によって異なる

$k = 0$ のとき				$k = 1$ のとき				注釈
13.	x	$\leftarrow$	$C_{00} - C_{30}$	x	$\leftarrow$	$C_{20} - C_{10}$		
14.	x	$\leftarrow$	$x \times C_{10}$	x	$\leftarrow$	$x \times C_{10}$		
15.	y	$\leftarrow$	$C_{01} - C_{31}$	y	$\leftarrow$	$C_{21} + C_{31}$		
16.	y	$\leftarrow$	$y \times C_{11}$	y	$\leftarrow$	$y \times C_{11}$		
17.	T_{30}	$\leftarrow$	$x - y$	T_{30}	$\leftarrow$	$x - y$		
18.	T_{30}	$\leftarrow$	$T_{30} + C_{20}$	T_{30}	$\leftarrow$	$T_{30} + C_{00}$		
19.	x	$\leftarrow$	$C_{01} + C_{31}$	x	$\leftarrow$	$C_{21} - C_{31}$		
20.	x	$\leftarrow$	$x \times C_{10}$	x	$\leftarrow$	$x \times C_{10}$		
21.	y	$\leftarrow$	$C_{00} + C_{30}$	y	$\leftarrow$	$C_{20} + C_{30}$		
22.	y	$\leftarrow$	$y \times C_{11}$	y	$\leftarrow$	$y \times C_{11}$		
23.	T_{31}	$\leftarrow$	$x + y$	T_{31}	$\leftarrow$	$x + y$		
24.	T_{31}	$\leftarrow$	$T_{31} - C_{21}$	T_{31}	$\leftarrow$	$T_{31} - C_{01}$		$(T_{30}, T_{31}) = T[3]$
25.	C_{00}	$\leftarrow$	$T_{10} + 0$					
26.	C_{01}	$\leftarrow$	$T_{11} + 0$	$C[0] \leftarrow T[1]$				
27.	C_{10}	$\leftarrow$	$T_{30} + 0$					
28.	C_{11}	$\leftarrow$	$T_{31} + 0$	$C[1] \leftarrow T[3]$				
29.	C_{20}	$\leftarrow$	$T_{20} + 0$					
30.	C_{21}	$\leftarrow$	$T_{21} + 0$	$C[2] \leftarrow T[2]$				

表 5.2: d から d_m を計算するのに必要な演算の回数

	乗算	加減算	シフトによるべき乗
定理 3.9 (i)	$\lvert 3^n \rvert + \alpha - 1$	4	1
定理 3.9 (ii), アルゴリズム 3.2-1	10	6	2
定理 3.9 (ii), アルゴリズム 3.2-3	$8(\lvert m \rvert - 2)$	$22(\lvert m \rvert - 2)$	0
定理 3.9 (iii)	0	1	0
合計	$8\lvert m \rvert + \lvert 3^n \rvert + \alpha - 7$	$22\lvert m \rvert - 33$	3

ループが終了すると，m が偶数ならば $(C_{00}, C_{01}) = c_m$，m が奇数ならば $(C_{10}, C_{11}) = c_m$ となっている．

最後に，定理 3.9 のステップ (iii) の $d_m = c_m + c_m^q$ を計算するが，$c_m = (c_{m0}, c_{m1}) = c_{m0} + c_{m1}\sqrt{-1}$ とすると，$c_m^q = c_{m0} - c_{m1}\sqrt{-1}$ であるから，

$$d_m \leftarrow c_{m0} + c_{m0}$$

を行えば良い．

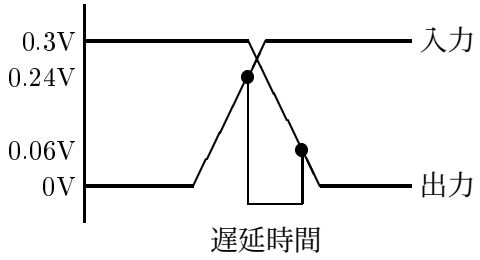
$(3^n + 1)/4$ を 2 進表現したときの 1 の個数を α とすると，d から d_m を求めるのに必要な演算の回数は表 5.2 のようになる．

5.3 標数 3 の体の XTR のためのハードウェアの設計と性能評価

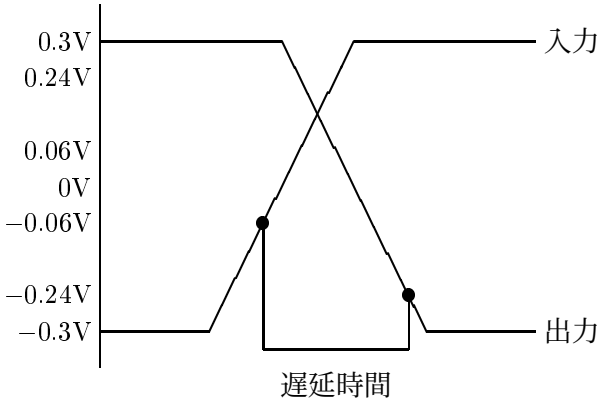
以上から標数 3 の体の XTR のための暗号ハードウェアに必要な要素は，標数 3 の体の加減算器，乗算器，べき乗計算のためのシフタ，パイプライン・ラッチやレジスタ，キューのためのラッチである．この節では，各要素をトランジスタレベルで設計し，HSPICE によるシミュレーションによって性能を評価する．なおパラメータは §4.6 で定めた値を用い，論理値 -1 には -0.3V，0 には 0V，1 には 0.3V が対応している．

5.3.1 遅延時間の測定の方針

ハードウェアの性能は処理にかかる時間によって評価されるので，ハードウェアの性能評価にとって遅延時間の測定は重要である．例えば電源電圧が 0.3V の 2 値論理では，0.06V までが論理値 0 を表すとし，0.24V 以上を論理値 1 を表すとすると，入力が $0 \to 1$ に変化して，出力が $1 \to 0$ に変化するようなハードウェアでは，入力が 0.24V になる時間から出力が 0.06V になるまでの時間を遅延時間とする．



本論文では，-0.24V 以下は論理値 -1，-0.06V から 0.06V は論理値 0，0.24V 以上は論理値 1 を表すとして，3 値論理のハードウェアの遅延時間を測定した．入力の論理値が 1 しか動かない場合は 2 値論理と同様の測定法で良いが，$-1 \to 1$ や $1 \to -1$ のように動く値が 2 の場合は，入力の値が 1 動いた時間から出力が確定するまでの時間を遅延時間とした．



5.3.2 加減算器

標数 3 の体の加減算器は，図 5.3 のように ADD ゲート，NOT ゲート，$T_{\overline{0}}$ ゲートから構成される．ADD ゲートを TG 法で構成すると図 5.7 のようになり，Olson 法で構成すると図 5.8 のようになる．トランジスタ数は，T ゲートは 16，$(0, 1, -1)$ ゲートは 6，$(1, -1, 0)$ ゲートは 6 なので，TG 法による ADD ゲートのトランジスタ数は 28 であり，Olson 法による ADD ゲートのトランジスタ数は 32 である．

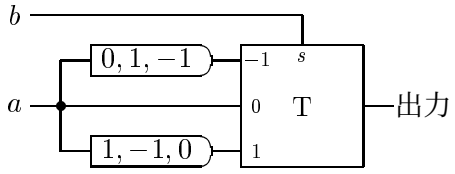


図 5.7: TG 法による ADD ゲート (入力は a と b)

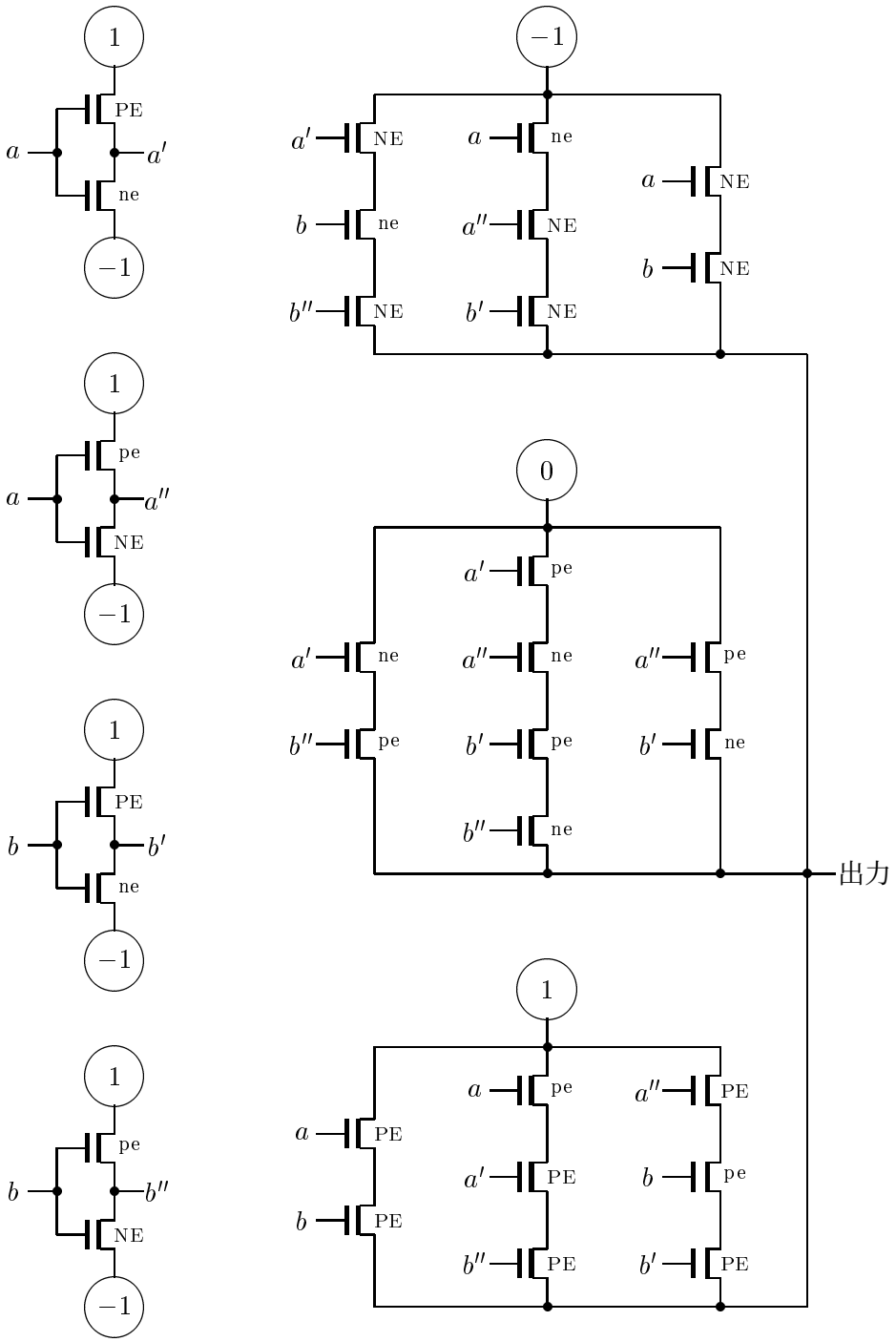


図 5.8: Olson 法による ADD ゲートの一例 (入力は a と b)

表 5.3: ADD ゲートの比較

	トランジスタ数	最大遅延
TG 法	28	2.2ns
Olson 法	32	3.0ns

TG 法の ADD ゲートは，入力の変化 $(-1,1) \to (-1,-1)$ に対して，出力が $0 \to 1$ と変化するときの遅延が最大となり (図 5.9)，Olson 法の ADD ゲートは，入力の変化 $(1,0) \to (0,0)$ に対して，出力が $1 \to 0$ と変化するときの遅延が最大となった (図 5.10)．2 つの構成法の ADD ゲートの比較は表 5.3 のようになり，TG 法の ADD ゲートが優れている．

この節の以降では 2 変数関数ゲートには TG 法を使っている．加減算器は，4 つの値 $\pm a \pm b$ を

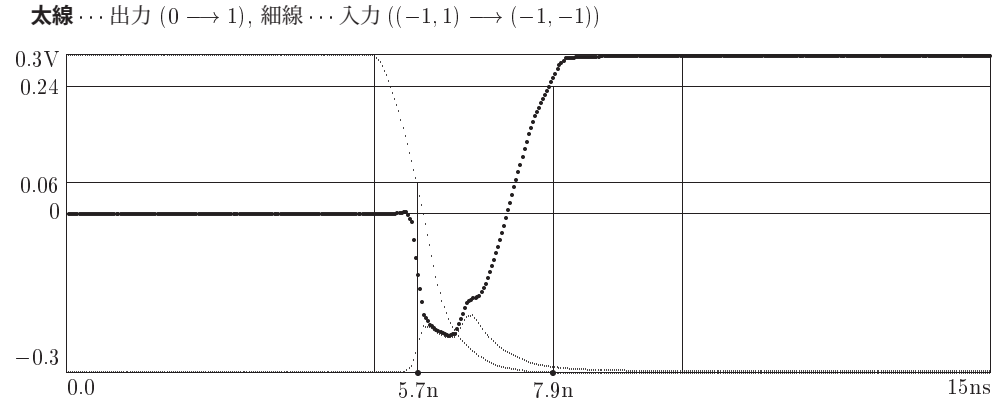


図 5.9: TG 法による ADD ゲートの最大遅延

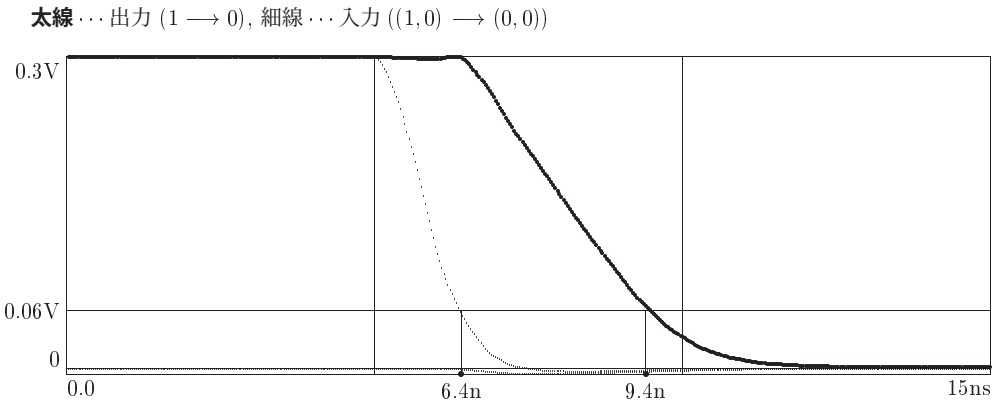


図 5.10: Olson 法による ADD ゲートの最大遅延

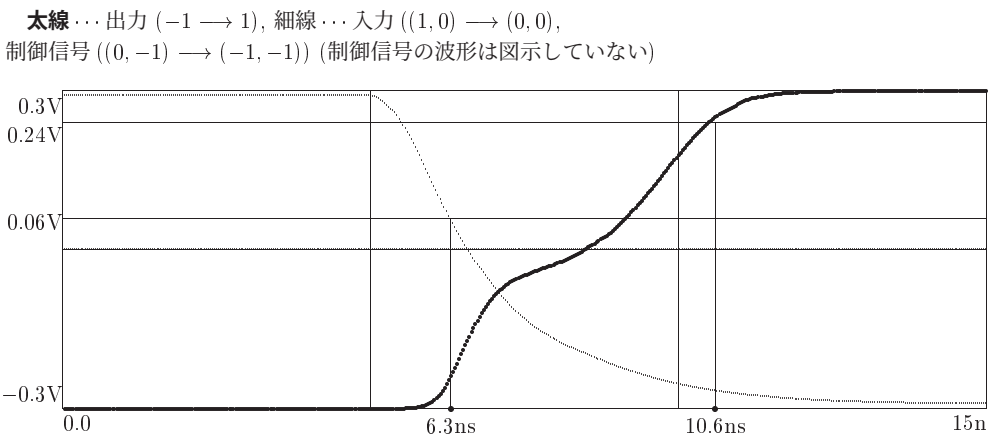


図 5.11: 加減算器の最大遅延

表 5.4: 1 桁の加減算器

最大遅延	ADD ゲートの個数	NOT ゲートの個数	$T_{\overline{0}}$ゲートの個数	トランジスタ数
4.3ns	4	3	3	164

計算し，2 つの制御信号 s_0, s_1 により 1 つの値を選択する回路である．加減算器は，入力の変化 $(1,0) \to (0,0)$ と制御信号の変化 $(0,-1) \to (-1,-1)$ に対して，出力が $-1 \to 1$ と変化するとき遅延時間が最大の 4.3ns となる (図 5.11, 表 5.4).

5.3.3 乗算器

標数 3 の体の乗算器は，部分積を生成する PRO ゲートからなる部分と，部分積を足し合わせるための ADD ゲートからなる部分とがある (図 5.4)．PRO ゲートは図 5.12 のようになり，最大遅延は 1.7ns である (図 5.14)．また，NOT のトランジスタ数は 4，T ゲートのトランジスタ数は 16 なので，PRO ゲートのトランジスタ数は 20 である (表 5.5).

単独の ADD ゲートについては図 5.9 と表 6.3 を参照．ここでは図 5.14 のような 3 つの ADD

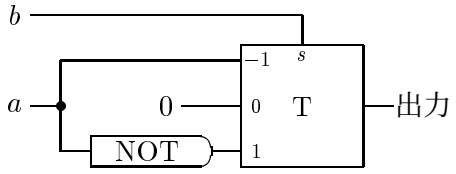


図 5.12: PRO ゲートの構成

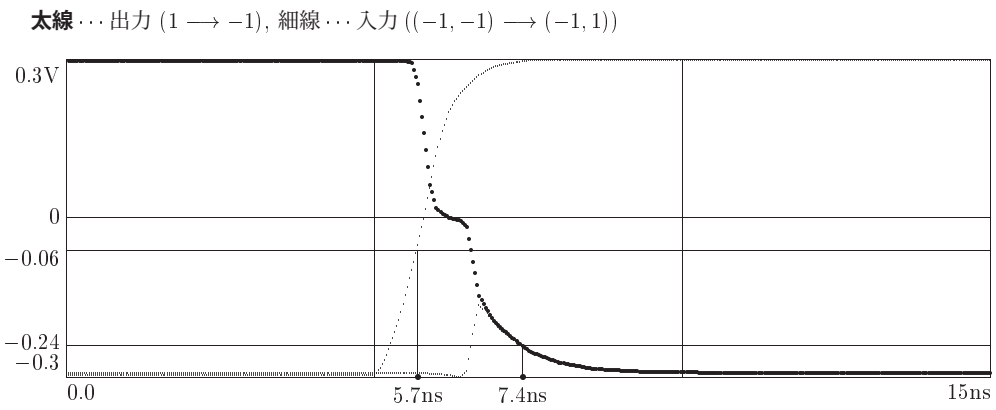


図 5.13: PRO ゲートの最大遅延

表 5.5: PRO ゲートの最大遅延とトランジスタ数

最大遅延	トランジスタ数
1.7ns	20

ゲートで構成される回路について調べる．このような回路は $\mathbb{F}_3$ の元 4 つの和を行い，これを 1 つのコンポーネントとみなすと，乗算器の部分積を足し合わせる部分の段数を削減できる．この回路の最大遅延は 3.9ns である (図 5.15)．

標数 3 の体の乗算器に関して，もう一つ重要なことは，体が $\mathbb{F}_{3^n}$ のときでは，部分積の値を $2n$ 個に分配しなければならないことである．このためには偶数 (約 $\log_2 n$) 段の逆二分木状の NOT ゲートを用いる (図 5.16)．$n = 111$ のときのこのツリー状の NOT ゲートの最大遅延は 3.4ns である (図 5.17)．

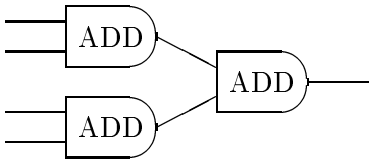


図 5.14: $\mathbb{F}_3$ の元 4 つの和を求める回路

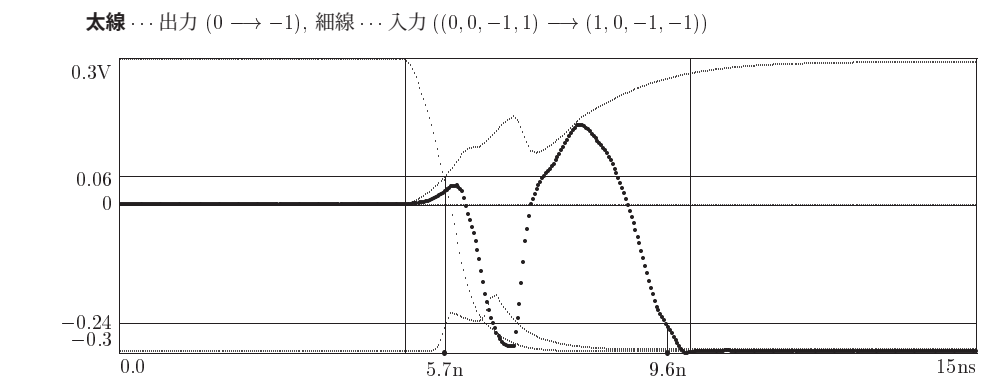


図 5.15: $\mathbb{F}_3$ の元 4 つの和を求める回路の最大遅延

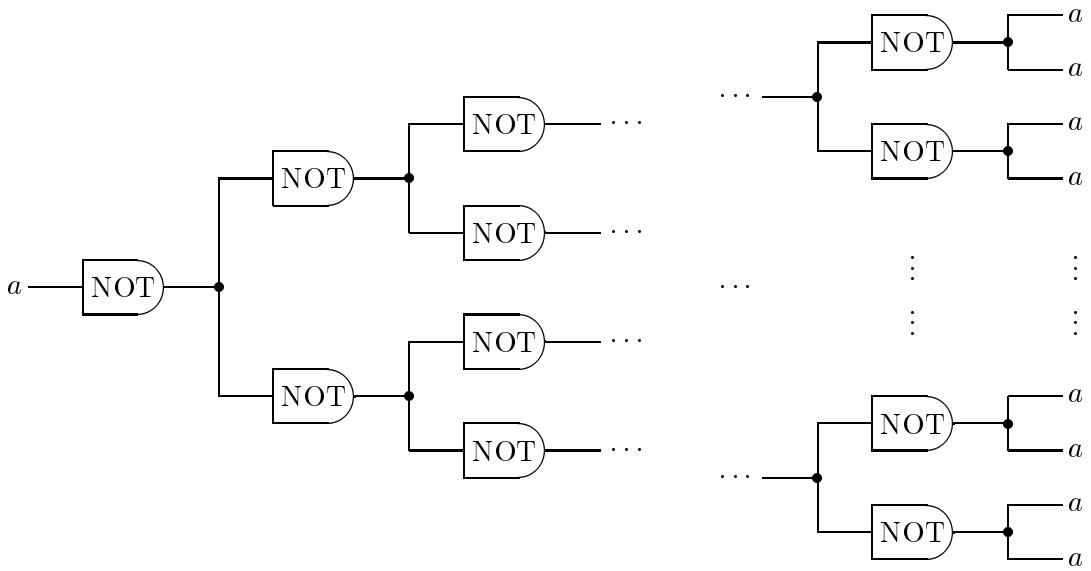


図 5.16: 信号を分配するためのツリー状の NOT ゲート

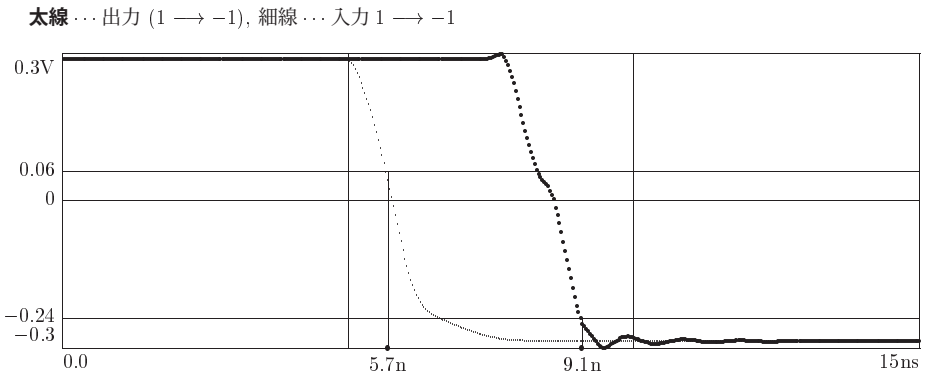


図 5.17: 信号を 222 個に分配するためのツリー状の NOT ゲートの最大遅延

5.4 ラッチ

ラッチは，クロックの立ち上がり時に入力値を取り入れ，クロックの立ち下がり時に取り入れた値を出力する．設計したラッチの波形の一例は図 5.18 のようになる．シミュレーションの結果により，クロックが立ち上がっていなければならない時間 (ラッチのセットアップ時間) は約 1ns, クロックが立ち下がってから出力値が正しい値になるまでの時間 (ラッチの遅延時間) も約 1ns を必要とすることが分かった．これはクロックサイクルを，各コンポーネントの遅延時間に 2ns を足したものにしなければならないことを意味する．

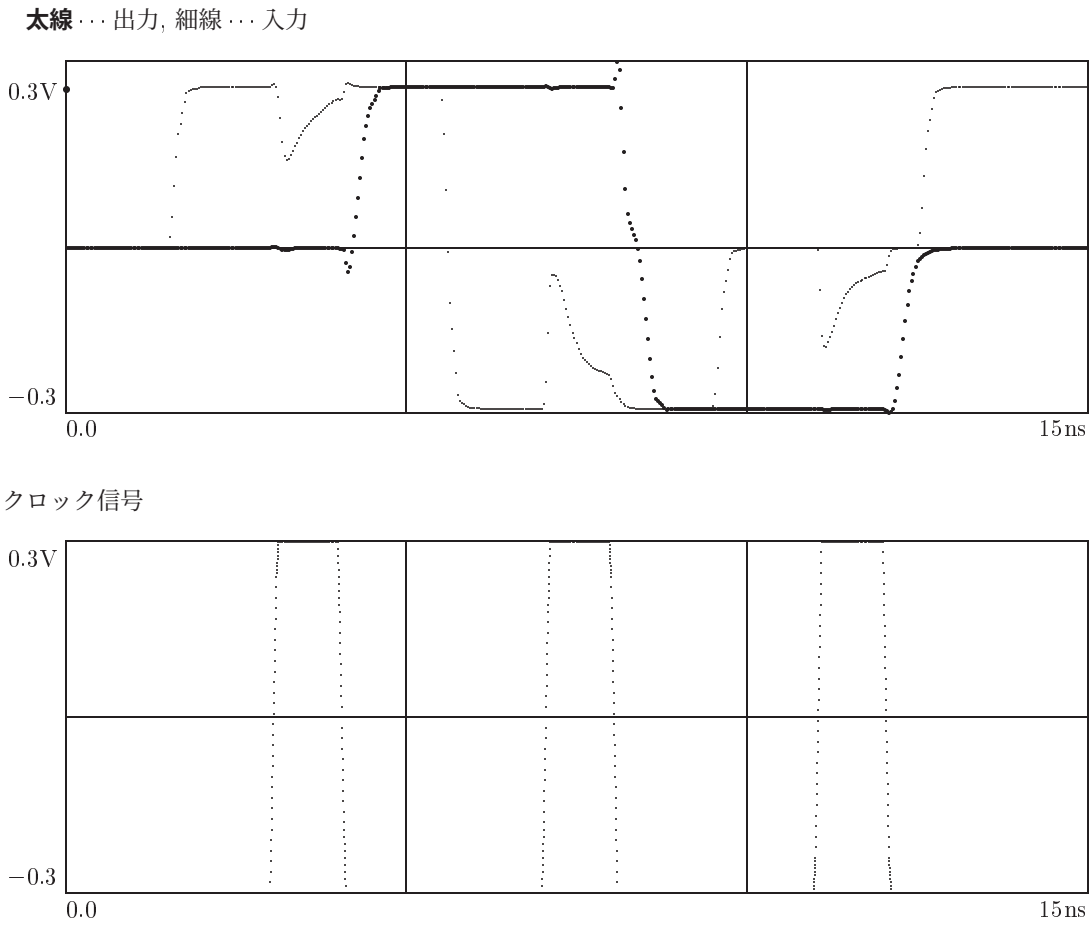


図 5.18: ラッチの波形の一例

5.5 標数 3 の体の XTR のためのハードウェアの性能評価

体として $\mathbb{F}_{3^{111}}$ を用いるときの XTR のためのハードウェアの性能を考える．実際には命令 ROM からの命令の読み出しに必要な時間も考慮しなければならないが，ここでは演算器の遅延時間によって暗号ハードウェアの性能を評価する．演算器の，加減算，部分積の生成，部分積の 222 の分配，部分積の 4 つの足し合わせ，に対してパイプライン分けする．これらに対する最大遅延時間は

加減算	部分積の生成	部分積の分配	部分積の 4 つの足し合わせ
4.3ns	1.7ns	3.4ns	3.9ns

である．ラッチのセットアップ時間が 1ns，ラッチの遅延時間が 1ns であるから，クロックサイクルは $4.3 + 2 = 6.3$ns となるが，配線での遅延やゆとりを考慮して，クロックサイクルは 10ns であるとする．すると，加減算には 1 クロック，乗算には部分積の生成に 1 クロック，部分積の分配に 1 クロック，部分積の足し合わせに 4 クロック，合計 6 クロックが必要である．(部分積の足し合わせでは，222 個の値を足し合わせ，また $4^3 < 222 < 4^4$ であるから,「部分積の 4 つの足し合わせ」が 4 段必要である．) シフタによるべき乗計算も 1 クロックでできるとする．

$d \in \mathbb{F}_{3^{111}}$ と整数 m から $d_m \in \mathbb{F}_{3^{111}}$ を計算することが，標数 3 の体の XTR の処理であるが，この処理のために必要なクロック数は表 5.2 から求めることができる．m は 1 から $(3^{111} - \sqrt{3^{112}} + 1$ の最大素因数) から選ばれるが，この素数は 156 ビットなので，$|m| = 156$ である．また $|3^{111}| = 176$ である．α は $(3^{111} + 1)/4$ を 2 進表現で表したときの 1 の個数であり，84 である．よって

$$\begin{aligned} 8|m| + |3^{111}| + \alpha - 7 &= 1501 \\ 22|m| - 18 &= 3399 \end{aligned}$$

であるから，必要なクロック数は $1501 \times 6 + 3399 \times 1 + 3 \times 1 = 12408$ クロックであり，この時間は 124.08μs である．なお，今回のシミュレーションでは電源電圧に ± 0.3V を用いたが，遅延時間は電源電圧に反比例するため，現在の汎用プロセッサと同様に ± 1V にすると，より高性能を望むことができる．

5.6 3 値論理と 2 値論理との比較

2 値論理で標数 3 の体を実装する場合は，$\mathbb{F}_3$ の元を 1 つ表現するのに，2 ビットが必要になる．$\mathbb{F}_3$ の元 a に対して，

a	$[a_1, a_0]$
-1	[1,0]
0	[0,0]
1	[0,1]

のように対応させるとする．3 値論理の NOT，ADD，PRO ゲートを 2 値論理ゲートで構成するとどうなるかを論ずる．

5.6.1 NOT ゲート

$b = \mathrm{NOT}(a)$ の関係を 2 値論理で表すと

$a = [a_1, a_0]$	$b = [b_1, b_0]$
$-1 = [1, 0]$	$1 = [0, 1]$
$0 = [0, 0]$	$0 = [0, 0]$
$1 = [0, 1]$	$-1 = [1, 0]$

となる．よって，b_1 と b_0 は 2 値の論理式で

$$\begin{aligned} b_1 &= \overline{a_1} \cdot a_0 = \overline{(a_1 + \overline{a_0})} \\ b_0 &= a_1 \cdot \overline{a_0} = \overline{(\overline{a_1} + a_0)} \end{aligned}$$

と表される．NOT ゲートは 2 値論理インバータ 2 つと 2 入力の 2 値論理 NOR ゲートで生成することができる．2 値論理インバータは 2 つの MOSFET で構成され，2 入力の NOR ゲートは 4 つの MOSFET で構成されるので，2 値論理で NOT ゲートを構成すると，$2 \times 2 + 2 \times 4 = 12$ 個の MOSFET が必要になる．3 値論理を用いると，NOT ゲートに必要な MOSFET 数は 4 である．

5.6.2 ADD ゲート

$c = \mathrm{ADD}(a, b)$ とし，$a = [a_1, a_0], b = [b_1, b_0], c = [c_1, c_0]$ と表すとする．$[c_1, c_0]$ のカルノー図は

		$a = [a_1, a_0]$		
		$[1, 0]$	$[0, 0]$	$[0, 1]$
$b = [b_1, b_0]$	$[1, 0]$	$[0, 1]$	$[1, 0]$	$[0, 0]$
	$[0, 0]$	$[1, 0]$	$[0, 0]$	$[0, 1]$
	$[0, 1]$	$[0, 0]$	$[0, 1]$	$[1, 0]$

となる．よって c_1，c_0 は

$$\begin{aligned} c_1 &= \overline{a_1}\cdot\overline{a_0}\cdot b_1\cdot\overline{b_0} + a_1\cdot\overline{a_0}\cdot\overline{b_1}\cdot\overline{b_0} + \overline{a_1}\cdot\overline{a_0}\cdot b_1\cdot b_0 \\ &= \overline{\overline{(\overline{a_1}\cdot\overline{a_0}\cdot b_1\cdot\overline{b_0})}\cdot\overline{(a_1\cdot\overline{a_0}\cdot\overline{b_1}\cdot\overline{b_0})}\cdot\overline{(\overline{a_1}\cdot\overline{a_0}\cdot b_1\cdot b_0)}} \\ c_0 &= a_1\cdot\overline{a_0}\cdot b_1\cdot\overline{b_0} + \overline{a_1}\cdot a_0\cdot\overline{b_1}\cdot\overline{b_0} + \overline{a_1}\cdot\overline{a_0}\cdot b_1\cdot b_0 \\ &= \overline{\overline{(a_1\cdot\overline{a_0}\cdot b_1\cdot\overline{b_0})}\cdot\overline{(\overline{a_1}\cdot a_0\cdot\overline{b_1}\cdot\overline{b_0})}\cdot\overline{(\overline{a_1}\cdot\overline{a_0}\cdot\overline{b_1}\cdot b_0)}} \end{aligned}$$

と表される．c_1 を生成するには，4 つの 2 値論理のインバータと，3 つの 4 入力の 2 値論理 NAND ゲートと，1 つの 3 入力の 2 値論理 NAND ゲートが必要である．c_0 を生成するにも，4 つの 2 値論理のインバータと，3 つの 4 入力の 2 値論理 NAND ゲートと，1 つの 3 入力の 2 値論理 NAND ゲートが必要であるが，インバータは c_1 を生成するときに用いるものと共通にすることができる．よって，ADD を 2 値論理ゲートで生成するには，4 つの 2 値論理インバータ，6 個の 4 入力の 2 値論理 NAND ゲート，2 つの 3 入力の 2 値論理 NAND ゲートが必要である．2 値論理インバータは 2 つの MOSFET で，4 入力 NAND ゲートは 8 個の MOSFET で，3 入力の NAND ゲートは 6 個の MOSFET で構成されるから，2 値論理で ADD ゲートを構成するのに必要な MOSFET 数は $4\times2+6\times8+2\times6=68$ 個の MOSFET が必要である．なお，3 値論理では ADD ゲートは 28 個の MOSFET で構成できる．

5.6.3 PRO ゲート

$c=\mathrm{PRO}(a,b)$ とし，$a=[a_1,a_0], b=[b_1,b_0], c=[c_1,c_0]$ と表すと，$[c_1,c_0]$ のカルノー図は

		$a=[a_1,a_0]$		
		$[1,0]$	$[0,0]$	$[0,1]$
$b=[b_1,b_0]$	$[1,0]$	$[0,1]$	$[0,0]$	$[1,0]$
	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
	$[0,1]$	$[1,0]$	$[0,0]$	$[0,1]$

となる．よって c_1，c_0 は

$$\begin{aligned} c_1 &= \overline{a_1}\cdot a_0\cdot b_1\cdot\overline{b_0} + a_1\cdot\overline{a_0}\cdot\overline{b_1}\cdot b_0 \\ &= \overline{\overline{(\overline{a_1}\cdot a_0\cdot b_1\cdot\overline{b_0})}\cdot\overline{(a_1\cdot\overline{a_0}\cdot\overline{b_1}\cdot b_0)}} \\ c_0 &= a_1\cdot\overline{a_0}\cdot b_1\cdot\overline{b_0} + \overline{a_1}\cdot a_0\cdot\overline{b_1}\cdot b_0 \\ &= \overline{\overline{(a_1\cdot\overline{a_0}\cdot b_1\cdot\overline{b_0})}\cdot\overline{(\overline{a_1}\cdot a_0\cdot\overline{b_1}\cdot b_0)}} \end{aligned}$$

となる．c_1 と c_0 を生成するのに必要な 2 値論理インバータを共通化すると，PRO ゲートは 4 つの 2 値論理インバータ，4 つの 4 入力の 2 値論値 NAND ゲート，2 つの 2 入力の 2 値論理 NAND ゲートによって構成でき，MOSFET 数は 4×2(インバータの MOSFET 数) $+\ 4\times$

8(4 入力 NAND の MOSFET 数) + 2 × 4(2 入力 NAND の MOSFET 数) = 48 である．なお，3 値論理では PRO ゲートの MOSFET 数は 20 である．

5.6.4　3 値論理と 2 値論理との比較

標数 3 の体での XTR のハードウェアの演算器に必要なゲートは NOT ゲート，ADD ゲート，PRO ゲートであり，これらを 3 値論理ゲートと 2 値論理ゲートで構成するときに必要な MOSFET 数は

	3 値論理	2 値論理
NOT	4	12
ADD	28	68
PRO	20	48

である．また，$\mathbb{F}_3$ の元 1 つを表現するときに必要な桁数は，3 値論理では 1，2 値論理では 2 である．よって，3 値論理を用いると，標数 3 の体での XTR のハードウェアの面積は半分以下にできることが期待できる．

5.7　5 章のまとめ

5 章では，TG 法による 3 値論理ゲートの遅延時間の HSPICE のシミュレーション結果から，標数 3 の体の XTR のためのハードウェアの性能を見積もった．TG 法によるゲートの遅延時間は数 ns であった．しかし，2 値論理のゲートの遅延時間はおよそ数十 ps であるので，電源電圧の大きさを考慮しても，TG 法による 3 値論理ゲートは性能が良いとは言えない．よって，XTR のためのハードウェアの性能もまだ十分ではない．

しかしながらこの事実は悲観することではなく，今後の性能向上が期待できることを意味している．計算機の遅延時間は，トランジスタでの遅延時間と配線での遅延時間の和である．トランジスタでの遅延時間は，トランジスタのサイズに反比例するため，微細化を進めることで削減されてきた．近年の計算機の性能向上は，微細加工技術の進展による割合が大きい．これに対して，配線も微細化により長さは短くなるものの断面積も小さくなるため，配線での遅延時間はあまり削減されていない．以前はトランジスタでの遅延時間が配線での遅延時間よりはるかに大きかったため，微細化による性能向上が容易であったが，最近はあまり差が無くなってきたため，微細化による性能向上が頭打ちになりつつある．しかし，3 値論理ゲートでは，配線量の削減により面積が縮小することと，微細化により素子配線が縮小することにより，遅延が短縮できるので，配線遅延時間を削減できることが期待できる．

第 6 章

対称 3 進表現を用いた加算器と乗算器

この章では，加算器と乗算器に対称 3 進表現を用いることを提案する．ここで扱う加算器は桁上げ先見加算器であり，乗算器は Wallace tree 乗算器である．これらは論理段数がビット数の対数に比例するため高性能である．一般に Wallace tree 乗算器は配線が複雑になり多ビットの Wallace tree 乗算器の実装は困難であった．この章では Wallace tree の複雑さをノード数と配線数と配線の交差数で評価し，2 進表現の場合と対称 3 進表現の場合で比較する．Wallace tree 乗算器は対称 3 進表現によって構成すると非常に簡略化されることが分かる．なお，2 進表現での従来の加算器や乗算器は [26] に詳しく書かれている．

6.1 対称 3 進表現

公開鍵暗号に標数 3 の体を用いることを成案することは本論文の目的のひとつであるが，もう一つ目的が，対称 3 進表現を用いた暗号演算を論ずることである．対称 3 進表現を用いるとビット数の大きな乗算器が簡単に構成できることが長所の 1 つである．本節では対称 3 進表現を説明する．対称 3 進表現による演算器を実装するには 3 値論理が適しているが，標数 3 の体の実装にも適している．近年，楕円曲線暗号 [10, 27] に標数 3 の体を用いることが提案されている [3, 4, 5]．本論文では XTR に標数 3 の体を用いることを提案する．

3 進表現とは $\{0, 1, 2\}$ を用いた整数の表現であり，$(x_n x_{n-1} \cdots x_0)_3,\ x_i \in \{0, 1, 2\}$ と表し，この値は

$$x_n \cdot 3^n + x_{n-1} \cdot 3^{n-1} + \cdots + x_0 \cdot 3^0$$

を表している．例えば，

$$(2, 1, 0, 0, 1, 2)_3 = 2 \cdot 3^5 + 1 \cdot 3^4 + 0 \cdot 3^3 + 0 \cdot 3^2 + 1 \cdot 3^1 + 2 \cdot 3^0 = 572$$

である．n 桁の 3 進表現は，0 から $2(1 + 3 + \cdots + 3^{n-1}) = 3^n - 1$ までの整数を表現できる．

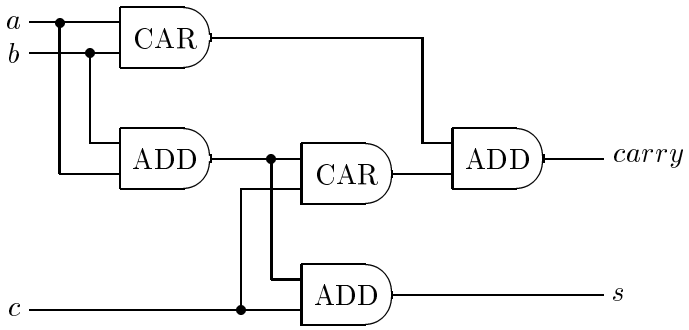


図 6.1: 対称 3 進表現での全加算器 FA

対称 3 進表現とは $\{-1,0,1\}$ を用いた 3 進表現であり，$(x_n x_{n-1}\cdots x_0)_3$，$x_i \in \{-1,0,1\}$ と表し，この値は

$$x_n \cdot 3^n + x_{n-1}\cdot 3^{n-1} + \cdots + x_0 \cdot 3^0$$

を表している．例えば,

$$(1,0,-1,-1,0,1)_3 = 1\cdot 3^5 + 0\cdot 3^4 - 1\cdot 3^3 - 1\cdot 3^2 + 0\cdot 3^1 + 1\cdot 3^0 = 208$$

である．n 桁の対称 3 進表現は，$-(1+3+\cdots+3^{n-1}) = -(3^n-1)/2$ から $(1+3+\cdots+3^{n-1}) = (3^n-1)/2$ までの整数を表現できる．

現在の一般的なコンピュータは，整数を表現するのに 2 進表現が用いられているが，プロセッサの配線や (暗号の演算に不可欠な) 乗算器を考えると 3 進表現を用いたほうが有利である．

6.2 対称 3 進表現での加算器

対称 3 進表現での全加算器 (FA) は図 6.1 のようになる．特に

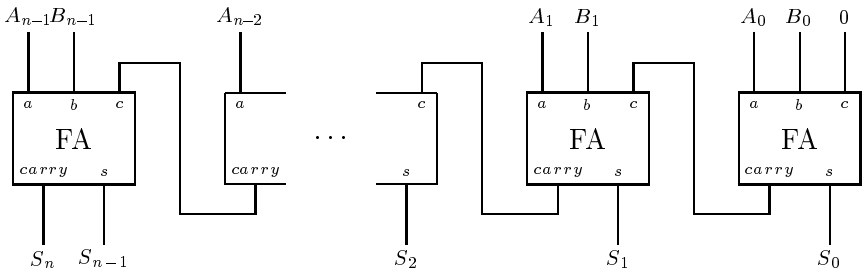
$$carry = a \odot b + c \odot (a+b) \tag{6.1}$$

となることが分かる．ここで，$\odot$ は CAR 関数，$+$ は ADD 関数である．2 進表現の場合のキャリー生成は

$$carry = a\cdot b + c\cdot (a+b) \tag{6.2}$$

である．(6.2) では $\cdot$ は AND，$+$ は OR である．関数は異なっているが，(6.1) と (6.2) は類似している．

$n-1$ 個の FA を



と並べれば n 桁の対称 3 進の加算器を構成できる．しかし，論理段数が $n-1$ に比例してしまうので，性能を求めるならば，桁上げ先見加算器が必要である．

初めに 2 進表現の桁上げ先見器を簡単に説明する．$A=(A_{n-1}A_{n-2}\cdots A_0)_2$ と $B=(B_{n-1}B_{n-2}\cdots B_0)_2$ を n ビットの加算したい値とし，C_i を i ビットで生じるキャリーとする．すると (6.1) より

$$C_i = A_i \cdot B_i + C_{i-1} \cdot (A_i + B_i) \tag{6.3}$$

となる．$G_i = A_i \cdot B_i$，$T_i = A_i + B_i$ とおくと，(6.3) は

$$C_i = G_i + T_i \cdot C_{i-1} \tag{6.4}$$

となる．C_i を展開していくと

$$\begin{aligned} C_i &= G_i + T_i \cdot C_{i-1} \\ &= G_i + T_i \cdot (G_{i-1} + T_{i-1} \cdot C_{i-2}) \\ &= G_i + T_i \cdot G_{i-1} + T_i \cdot T_{i-1} \cdot C_{i-2} \\ &\quad\vdots \\ &= G_i + T_i \cdot G_{i-1} + \cdots + T_i \cdot T_{i-1} \cdot \cdots \cdot T_2 \cdot T_1 \cdot C_0 \end{aligned} \tag{6.5}$$

となり，各キャリー C_i は各 A_i, B_i と C_0 の論理から生成できる．ところで，(6.5) のように展開できるのは AND と OR が分配法則

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

を満たすからである．

次に対称 3 進表現での桁上げ先見器を考える．n 桁の対称 3 進表現の整数 $A=(A_{n-1}A_{n-2}\cdots A_0)_3$ と $B=(B_{n-1}B_{n-2}\cdots B_0)_3$ の加算で生じる i 桁目のキャリーを C_i とする．2 進表現のときと同様に $G_i = A_i \odot B_i$, $T_i = A_i + B_i$ とおくと (6.1) より

$$C_i = G_i + T_i \odot C_{i-1} \tag{6.6}$$

が得られる．ここで問題となるのは，CAR$\odot$ と ADD+ が分配法則

$$a \odot (b + c) = a \odot b + a \odot c$$

を満たさないので，(6.5) のように (6.6) を式展開できないことである．

幸いにも，PRO $\cdot$ と ADD $+$ は分配法則を満たすので，$\odot$ を $\cdot$ と $+$ で表現できれば良い．

$$a \odot b = -a^2 \cdot b - a \cdot b^2$$

であることがカルノー図

$a^2 \cdot b$ の値

b \ a	-1	0	1
-1	-1	0	-1
0	0	0	0
1	1	0	1

$a \cdot b^2$ の値

b \ a	-1	0	1
-1	-1	0	1
0	0	0	0
1	-1	0	1

$a \odot b = -a^2 \cdot b - a \cdot b^2$ の値

b \ a	-1	0	1
-1	-1	0	0
0	0	0	0
1	0	0	1

から分かる．よって (6.6) は

$$C_i = G_i - T_i^2 \cdot C_{i-1} - T_i \cdot C_{i-1}^2$$

となる．右辺に C_{i-1}^2 があるので，(6.4) のような関係を得るためには C_i^2 が必要になる．$T_i = -1,\ 0,\ 1$ の場合に分けて，C_i^2 や後で必要になる値を計算する．なお，$a \in \{-1, 0, 1\}$ に対して，$a^3 = a,\ 2a = -a$ であることに注意．

- $T_i = -1$ のとき

$$\begin{aligned} C_i &= G_i - C_{i-1} + C_{i-1}^2 && (6.7)\\ C_i^2 &= G_i^2 + (G_i + 1) \cdot C_{i-1} + (2G_i + 2) \cdot C_{i-1}^2 \\ C_i + C_i^2 &= G_i + G_i^2 + G_i \cdot (C_{i-1} - C_{i-1}^2) && (6.8)\\ C_i - C_i^2 &= G_i - G_i^2 + (2G_i + 1) \cdot (C_{i-1} - C_{i-1}^2) && (6.9) \end{aligned}$$

- $T_i = 0$ のとき

$$\begin{aligned} C_i &= G_i && (6.10)\\ C_i^2 &= G_i^2 \\ C_i + C_i^2 &= G_i + G_i^2 && (6.11)\\ C_i - C_i^2 &= G_i - G_i^2 && (6.12) \end{aligned}$$

- $T_i = 1$ のとき

$$\begin{aligned} C_i &= G_i - C_{i-1} - C_{i-1}^2 && (6.13)\\ C_i^2 &= G_i^2 + (G_i - 1) \cdot C_{i-1} + (G_i - 1) \cdot C_{i-1}^2 \\ C_i + C_i^2 &= G_i + G_i^2 + (G_i + 1) \cdot (C_{i-1} + C_{i-1}^2) && (6.14)\\ C_i - C_i^2 &= G_i - G_i^2 + 2G_i \cdot (C_{i-1} + C_{i-1}^2) && (6.15) \end{aligned}$$

更に $T_{i+1} = -1, 0, 1$ の場合に分けて $C_i + T_{i+1} \cdot C_i^2$ を考える.

- $T_i = -1,\ T_{i+1} = -1$ のとき，(6.9) より

$$C_i + T_{i+1} \cdot C_i^2 = G_i + T_{i+1} \cdot G_i^2 + (2G_i + 1) \cdot (C_{i-1} + T_i \cdot C_{i-1}^2) \tag{6.16}$$

- $T_i = -1,\ T_{i+1} = 0$ のとき，(6.7) より

$$C_i + T_{i+1} \cdot C_i^2 = G_i + T_{i+1} \cdot G_i^2 - (C_{i-1} + T_i \cdot C_{i-1}^2) \tag{6.17}$$

- $T_i = -1,\ T_{i+1} = 1$ のとき，(6.8) より

$$C_i + T_{i+1} \cdot C_i^2 = G_i + T_{i+1} \cdot G_i^2 + G_i \cdot (C_{i-1} + T_i \cdot C_{i-1}^2) \tag{6.18}$$

- $T_i = 0,\ T_{i+1} = -1$ のとき，(6.12) より

$$C_i + T_{i+1} \cdot C_i^2 = G_i + T_{i+1} \cdot G_i^2 + 0 \cdot (C_{i-1} + T_i \cdot C_{i-1}^2) \tag{6.19}$$

- $T_i = 0,\ T_{i+1} = 0$ のとき，(6.10) より

$$C_i + T_{i+1} \cdot C_i^2 = G_i + T_{i+1} \cdot G_i^2 + 0 \cdot (C_{i-1} + T_i \cdot C_{i-1}^2) \tag{6.20}$$

- $T_i = 0,\ T_{i+1} = 1$ のとき，(6.11) より

$$C_i + T_{i+1} \cdot C_i^2 = G_i + T_{i+1} \cdot G_i^2 + 0 \cdot (C_{i-1} + T_i \cdot C_{i-1}^2) \tag{6.21}$$

- $T_i = 1,\ T_{i+1} = -1$ のとき，(6.15) より

$$C_i + T_{i+1} \cdot C_i^2 = G_i + T_{i+1} \cdot G_i^2 + 2G_i \cdot (C_{i-1} + T_i \cdot C_{i-1}^2) \tag{6.22}$$

- $T_i = 1,\ T_{i+1} = 0$ のとき，(6.13) より

$$C_i + T_{i+1} \cdot C_i^2 = G_i + T_{i+1} \cdot G_i^2 - (C_{i-1} + T_i \cdot C_{i-1}^2) \tag{6.23}$$

- $T_i = 1,\ T_{i+1} = 1$ のとき，(6.14) より

$$C_i + T_{i+1} \cdot C_i^2 = G_i + T_{i+1} \cdot G_i^2 + (G_i + 1) \cdot (C_{i-1} + T_i \cdot C_{i-1}^2) \tag{6.24}$$

よって，$D_i = C_i + T_{i+1} \cdot C_i^2$，$E_i = G_i + T_{i+1} \cdot G_i^2$ とおき，F_i を $T_i,\ T_{i+1}$ に従い

F_iの値

		T_i		
		-1	0	1
T_{i+1}	-1	$-G_i+1$	0	$-G_i$
	0	-1	0	-1
	1	G_i	0	G_i+1

とおくと，(6.16)〜(6.24) より $D_i,\ E_i,\ F_i$ は

$$D_i = E_i + F_i \cdot D_{i-1} \tag{6.25}$$

を満たすことが分かる．(6.25) は (6.5) のように展開できる．

$$\begin{aligned} D_i &= E_i + F_i \cdot D_{i-1} \\ &= E_i + F_i \cdot (E_{i-1} + F_{i-1} \cdot D_{i-2}) \\ &= E_i + F_i \cdot E_{i-1} + F_i \cdot F_{i-1} \cdot D_{i-2} \\ &\quad \vdots \\ &= E_i + F_i \cdot E_{i-1} + \cdots + F_i \cdot F_{i-1} \cdot \cdots \cdot F_2 \cdot F_1 \cdot D_0 \end{aligned} \tag{6.26}$$

各 $E_i,\ F_i$ は $A_i,\ B_i,\ C_0$ の論理で生成できるから，各 D_i も $A_i,\ B_i,\ C_0$ の論理で生成できる．しかし，これではキャリー C_i を生成したことにはならず，$D_i = C_i + T_{i+1} \cdot C_i^2$ から C_i を復元しなければならない．D_i の値はカルノー図

D_iの値

		C_i		
		-1	0	1
T_{i+1}	-1	1	0	0
	0	-1	0	1
	1	0	0	-1

で表される．この関係から，D_i を使って C_i を求めなければならない．

まず，C_i' を

$$C_i' = \begin{cases} 0 & D_i = 0 \text{ のとき} \\ D_i - T_{i+1} & D_i \neq 0 \text{ のとき} \end{cases}$$

と定義する．C'_i と，C_i, T_{i+1} との関係はカルノー図で表すと，

C'_iの値

		C_i		
		-1	0	1
T_{i+1}	-1	-1	0	$\mathbf{0}$
	0	-1	0	1
	1	$\mathbf{0}$	0	1

となる．2 つの太字の 0 のうち，上の方が 1，下の方が -1 になれば，C'_i と C_i は一致することになる．

$$C_i = C'_i + C''_i$$

とおいて，C''_i を求めることができれば，C_i が得られる．C''_i と，C_i, T_{i+1} との関係をカルノー図で表すと，

C''_iの値

		C_i		
		-1	0	1
T_{i+1}	-1	0	0	1
	0	0	0	0
	1	-1	0	0

である．

まず，$C''_i = 1$ となる条件を考える．$C''_i = 1$ となるのは，$T_{i+1} = -1$ かつ $C_i = 1$ のときである．また，$C_i = 1$ であることの必要十分条件は

$$A_i + B_i + C_{i-1} = A_i + B_i + C'_{i-1} + C''_{i-1} \geq 2 \tag{6.27}$$

である．実は条件 (6.27) は

$$A_i + B_i + C'_{i-1} \geq 2 \tag{6.28}$$

と同じであることを示す．$C''_{i-1} = 0$ のときは，(6.27) と (6.28) は一致する．

$C''_{i-1} = -1$ と仮定する．ことのとき $T_i = 1$ でなければならない．T_i の定義より $A_i + B_i = 1$ である．すると (6.27) は $1 + C'_{i-1} - 1 = C'_{i-1} \geq 2$ となるが，$C'_{i+1} \in \{-1, 0, 1\}$ であるから，このような場合は起こり得ない．

$C''_{i-1} = 1$ と仮定しても，同様である．ことのとき $T_i = A_i + B_i = -1$ でなければならないが，すると (6.27) は $-1 + C'_{i-1} + 1 = C'_{i-1} \geq 2$ となり，このような場合は起こり得ない．

よって条件 (6.27) と (6.28) は一致することが示せた．以上から

$$T_{i+1} = -1 \text{ かつ } A_i + B_i + C'_{i-1} \geq 2$$

のとき $C''_i = 1$ となる．つまり，C''_i の値は C''_{i-1} には関係しない．更に，$A_i + B_i + C'_{i-1} \geq 2$ であることと $A_i + B_i + C'_{i-1}$ が桁上がりすることは同じだから，

$$A_i + B_i + C'_{i-1} \geq 2 \Leftrightarrow A_i \odot B_i + C'_{i-1} \odot (A_i + B_i) = G_i + C'_{i-1} \odot T_i = 1$$

である．同様に

$$T_{i+1} = 1 \text{ かつ } G_i + C'_{i-1} \odot T_1 = -1$$

のとき $C''_i = -1$ となる．

以上から次のキャリー生成アルゴリズムが得られる．

アルゴリズム 6.1 キャリー生成アルゴリズム

n 桁の対称 3 進表現の整数 $A = (A_{n-1}A_{n-2}\cdots A_0)_3$ と $B = (B_{n-1}B_{n-2}\cdots B_0)_3$ の加算を行うときのキャリー C_i は次のようにして求められる．

1. $G_i = A \odot B_i,\ T_i = A_i + B_i$ とおく．また $T_n = 0$ とする．
2. $E_i = G_i + T_{i+1} \cdot G_i^2$ とおき，T_i と T_{i+1} の値により F_i を

F_iの値

T_{i+1} \ T_i	-1	0	1
-1	$-G_i + 1$	0	$-G_i$
0	-1	0	-1
1	G_i	0	$G_i + 1$

   とする．
3. $D_i = E_i + F_i \cdot D_{i-1}$ とおく．すると

$$D_i = E_i + F_i \cdot E_{i-1} + \cdots + F_i \cdot F_{i-1} \cdot \cdots \cdot F_2 \cdot F_1 \cdot D_0$$

   となり，各 D_i は各 $A_i,\ B_i,\ C_0$ の論理によって求められる．
4. C'_i を

$$C'_i = \begin{cases} 0 & D_i = 0 \text{ のとき} \\ D_i - T_{i+1} & D_i \neq 0 \text{ のとき} \end{cases}$$

   とする．

5. C_i'' を

$$C_i'' = \begin{cases} 1 & T_{i+1} = -1 \text{ かつ } G_i + C_{i-1}' \odot T_i = 1 \text{ のとき} \\ -1 & T_{i+1} = 1 \text{ かつ } G_i + C_{i-1}' \odot T_i = -1 \text{ のとき} \\ 0 & \text{その他のとき} \end{cases}$$

とする．

6. $C_i = C_i' + C_i''$ となる． □

このアルゴリズムは2進表現でのキャリー生成アルゴリズムとかなりに似いる．大きな違いステップ2，4，5が増えていることだが，これらはコストの低い行程である．ステップ3がこのアルゴリズムの中心となるが，この部分は関数は異なっているが，2進表現の場合と同じ手順であり，ステップ3の論理段数は n の対数に比例する．

6.3 乗算器の概要

この節では一般の (2進表現での) 乗算器を説明する．乗算 $a_3a_2a_1a_0 \times b_3b_2b_1b_0$ の筆算を考えると，部分積 a_ib_j の生成と加算を行っていることが分かる．

$$\begin{array}{cccccccc} & & & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\ & & & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\ & & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\ + & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & \\ \hline \end{array}$$

乗算器では，部分積を生成するコンポーネントを MPS(MultiPle Select) といい，加算を行うコンポーネントには CSA(Carry Save Adder) と CPA(Carry Propagate Adder) がある．MPS は1ビットの乗算器である AND ゲートにより構成される．各 $(a_3b_i, a_2b_i, a_1b_i, a_0b_i)$ を1つの値と考えると4つのデータの和をとることで乗算結果が得られる．n ビットの乗算では n 個のデータの和を行う．CSA は n 個のデータを2個のデータに減らすためのコンポーネントであり，2個のデータの和は普通の加算器 (CPA) で計算できる．CSA は全加算器 (full adder, FA) と半加算器 (half adder, HA) によって構成される．

単純な CSA の構成法としては，図 6.2 のような方法がある．ビット数を n とする．図 6.2 から分かるようにノード数 (FA と HA の個数) は

$$(n-1)^2 \tag{6.29}$$

である．図 6.2 のような CSA の構成ならば配置配線は簡単だが，CSA の論理段数は $n-1$ に比例し，乗算器の性能は良くない．

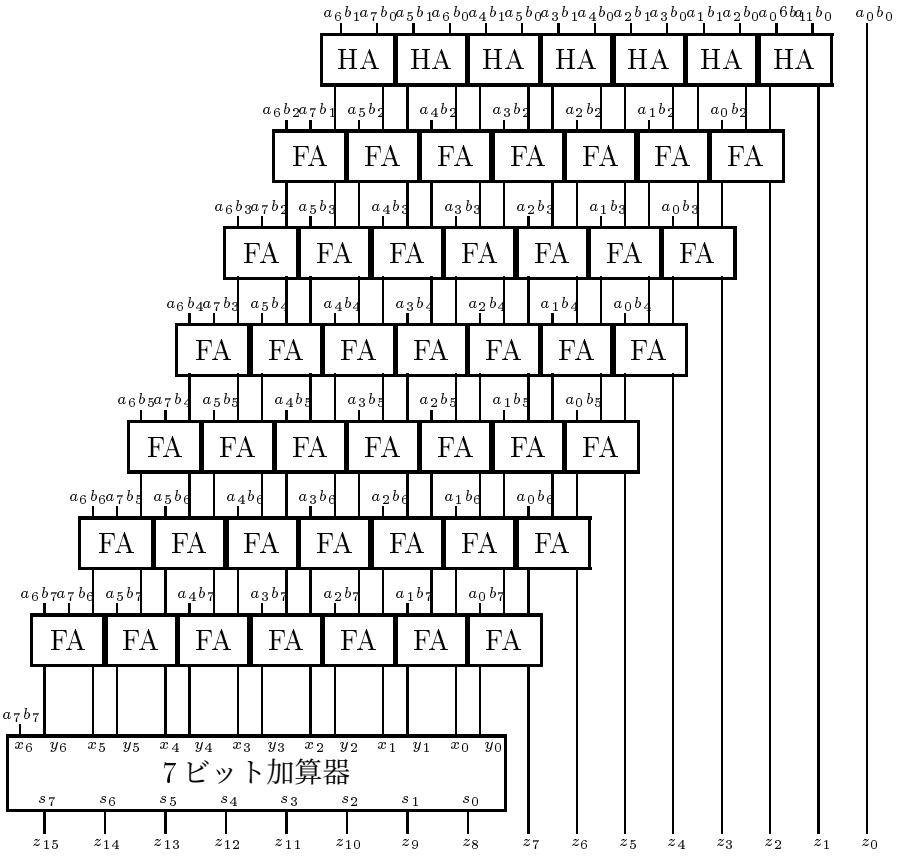


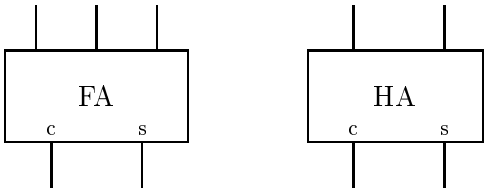
図 6.2: 8 ビット乗算器

CSA を Wallace tree にすることで，論理段数を $\log n$ に比例させることができる．ただし，n が大きくなると配置配線が非常に複雑になる．Wallace tree は本質的には図 6.2 のノードの並び替えであるので，必要なゲートは図 6.2 のような CSA でも Wallace tree でもほとんど同じである．

Wallace tree を簡単にするために対称 3 進表現を用いることが，この論文の主題の 1 つである．

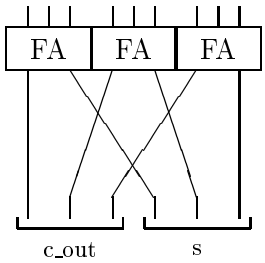
6.3.1 fa$n$

fa$n$ を定義すると，Wallace tree が考えやすくなる．FA と HA の出力を c(carry) 及び s(sum) と書くことにする．c は桁上り，s は和である．



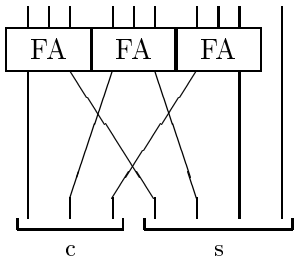
n ≡ 0 (mod 3) の時は全加算器を $n/3$ 個並べ，各全加算器の c 同士と s 同士をまとめたものが fa$n$ となる．c の集まりを fa$n$ の c，s の集まりを fa$n$ の s と呼ぶことにする．fa$n$ の c，s はそれぞれ $n/3$ ビットである．例えば fa9 は次のようになる．

fa9



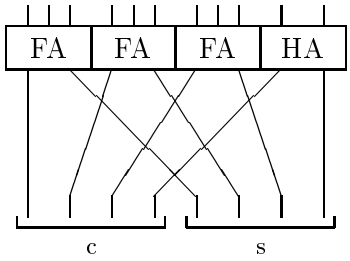
n ≡ 1 (mod 3) の時は全加算器を $\lfloor n/3 \rfloor$ 個並べその隣にスルー線を並べたものが fa$n$ となる．fa$n$ の c は各全加算器の c をまとめたもの，fa$n$ の s は各全加算器の s とスルー線をまとめたものとなる．fa$n$ の c は $\lfloor n/3 \rfloor$ ビット，fa$n$ の s は $\lfloor n/3 \rfloor + 1$ ビットとなる．ここで $\lfloor \cdot \rfloor$ は小数点以下の切り捨てを意味する．fa10 は次のようになる．

fa10



n ≡ 2 (mod 3) の時は全加算器を $\lfloor n/3 \rfloor$ 個と半加算器を 1 つ並べたものが fa$n$ となる．fa$n$ の c は各全加算器と半加算器の c をまとめたもの，fa$n$ の s は各全加算器と半加算器の s をまとめたものとなる．fa$n$ の c と s はともに $\lfloor n/3 \rfloor + 1$ ビットとなる．fa11 は次のようになる．

fa11



fa$n$ の論理段数は全加算器の論理段数と同じ 3 である．

6.3.2 配線の交差数

Wallace tree の構成には fa$n$ を用いることができるが，後に説明するように，fa$n$ 内の FA の s と c をまとめることで，fa$n$ 間の配線を交わらせること無く配置できるようになる．fa$n$ 内の s と c をまとめるため，fa$n$ 内では配線が交わっており，その個数を配線の交差数とよぶことにする．配線の交差数は配置配線の複雑さの規準の一つとなる．

fa$n$ の配線の交差数は fa$n$ の c が c ビットであるとすると

$$1 + 2 + \cdots + (c-1) = \frac{c(c-1)}{2} \tag{6.30}$$

である．

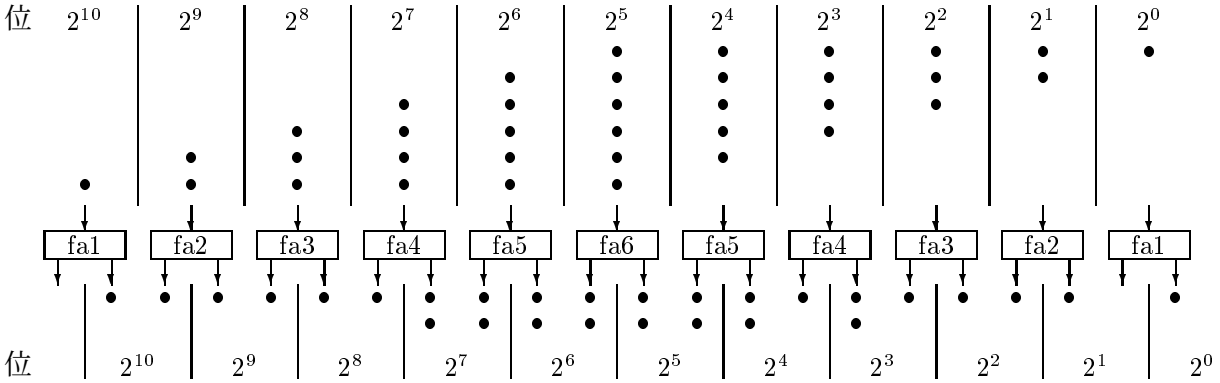
6.3.3 Wallace tree 乗算器

Wallace tree 乗算器は CSA 部を fa$n$ で構成した乗算器である．また 6 ビットの乗算を例にする．• をビット値とする．MPS 部の結果は次のようになる．

位	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
						•	•	•	•	•	•
					•	•	•	•	•	•	
				•	•	•	•	•	•		
			•	•	•	•	•	•			
		•	•	•	•	•	•				
	•	•	•	•	•	•					
• の個数	1	2	3	4	5	6	5	4	3	2	1

2^i の位のビット値が n 個あるならばビットを fa$n$ へ入力する．そうすると fa$n$ の出力のうち，c が 2^{i+1} の位のビット値，s が 2^i の位のビット値となる．

ステップ 0



ここまでの操作をステップ 0 とする．ステップ 0 の結果により次のようなビット値が得られる．

位	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	•	•	•	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•	•		
				•	•	•	•	•			
				•	•	•					
個数	2	2	2	4	4	4	3	3	2	1	1

ステップ 0 と同様にステップ 0 の結果の各位のビット値を fa$n$ に入力する．この操作をステップ 1 とする．同様にステップ 2 も行う．

ステップ 1 の結果=ステップ 2 の開始時

•	•	•	•	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•			
				•	•	•					
1	2	2	2	3	3	3	2	2	1	1	1

ステップ 2 の結果=CPA への入力時

•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•				
2	2	2	2	2	2	2	2	1	1	1	1

ステップ 2 が終わると各位のビット値が 2 つ以下になっているので，それらのビット値を CPA(8 ビット加算器) へ入力すれば 6 ビット乗算の結果が得られる．

- • の個数だけを書き出すと次のようになる．

ステップ 0 の開始時		1	2	3	4	5	6	5	4	3	2	1
ステップ 1 の開始時		2	2	2	4	4	4	3	3	2	1	1
ステップ 2 の開始時	1	2	2	2	3	3	3	2	2	1	1	1
CPA への入力時	2	2	2	2	2	2	2	2	1	1	1	1

この表から Wallace tree に必要な fa$n$ が分かる．

fa1	fa2	fa3	fa4	fa5	fa6	fa5	fa4	fa3	fa2	fa1
fa2	fa2	fa2	fa4	fa4	fa4	fa3	fa3	fa2	fa1	fa1
fa1	fa2	fa2	fa2	fa3	fa3	fa3	fa2	fa2	fa1	fa1	fa1

各 fa$n$ 間の配線を交差させないで，Wallace tree を配置配線することができる．よって Wallace tree の配線の交差数は各 fa$n$ 内の配線交差数の和となる．

k ビット Wallace tree 乗算器の CSA 部において，ステップ m での 2^i の位で使用する fa$n$ の n を $n_{m,i}$ と書くとする．$n_{m,i}$ は次のアルゴリズムで求めることができる．

アルゴリズム 6.2 fa$n$ の n を求める

1. $m = 0$ のとき (初期値設定)

$$\begin{aligned} i < k \quad &\Rightarrow \quad n_{0,i} = i + 1, \\ i \geq k \quad &\Rightarrow \quad n_{0,i} = 2k - 1 - i, \end{aligned}$$

2. $m \geq 1$ のとき

$n_{m,i}$ = fa $n_{m-1,i-1}$ の c のビット数 + fa $n_{m-1,i}$ の s のビット数. □

また，Wallace tree のノード数は各 fa$n$ の n の和になる．

6.4 対称 3 進表現での乗算と乗算器

乗算器を効率的に構成するには

1. 1 桁 ×1 桁 =1 桁
2. CSA に用いる n-2 Reducer の n の値が大きい

となるのが良い．もし 1 桁 ×1 桁 =2 桁になると，部分積の個数が 2 倍になり，CSA に必要な Reducer の個数も 2 倍になる．また，4-2 Reducer を用いて CSA を構成できるならば，CSA つまり Wallace tree の配線が簡単になる．対称 3 進表現での乗算はこれらの条件を満たしている．

6.4.1 1 桁 ×1 桁

対称 3 進表現と 3 進表現の 1 桁 ×1 桁は次のようになる．

対称 3 進表現での 1 桁乗算	3 進表現での 1 桁乗算
$-1 \times -1 = 1$	$0 \times 0 = 0$
$-1 \times 0 = 0$	$0 \times 1 = 0$
$-1 \times 1 = -1$	$0 \times 2 = 0$
$0 \times -1 = 0$	$1 \times 0 = 0$
$0 \times 0 = 0$	$1 \times 1 = 1$
$0 \times 1 = 0$	$1 \times 2 = 2$
$1 \times -1 = -1$	$2 \times 0 = 0$
$1 \times 0 = 0$	$2 \times 1 = 2$
$1 \times 1 = 1$	$2 \times 2 = 4(= (1,1)_3)$

よって対称 3 進表現では 1 桁 ×1 桁 =1 桁となるが，3 進表現では 1 桁 ×1 桁 =2 桁となる．

$n \geq 4$ では n 進表現でも対称 n 進表現でも 1 桁 ×1 桁 =2 桁となる．1 桁 ×1 桁 =1 桁となるのは，2 進表現と対称 3 進表現だけである．

6.4.2 Reducer について

CSA に n-2 Reducer が使用できるためには

$$\underbrace{1\text{ 桁} + \cdots + 1\text{ 桁}}_{n\text{ 個の和}} = 2\text{ 桁}$$

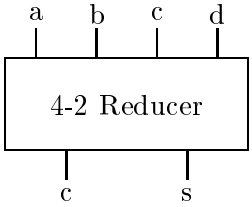
となることが必要である．対称 3 進表現では，$4 \times (-1) = -4 = (-1,-1)_3$，$4 \times 1 = 4 = (1,1)_3$ なので，

$$1\text{ 桁} + 1\text{ 桁} + 1\text{ 桁} + 1\text{ 桁} = 2\text{ 桁}$$

となる．対称 3 進表現では CSA に 4-2 Reducer を用いることができる．

6.4.3 対称 3 進表現での乗算器

対称 3 進表現では乗算器の CSA を 4-2 Reducer によって構成できる．ここでは対称 3 進表現を 3 値論理で実装するとする．3 値論理ゲートは 4 章で論じた．



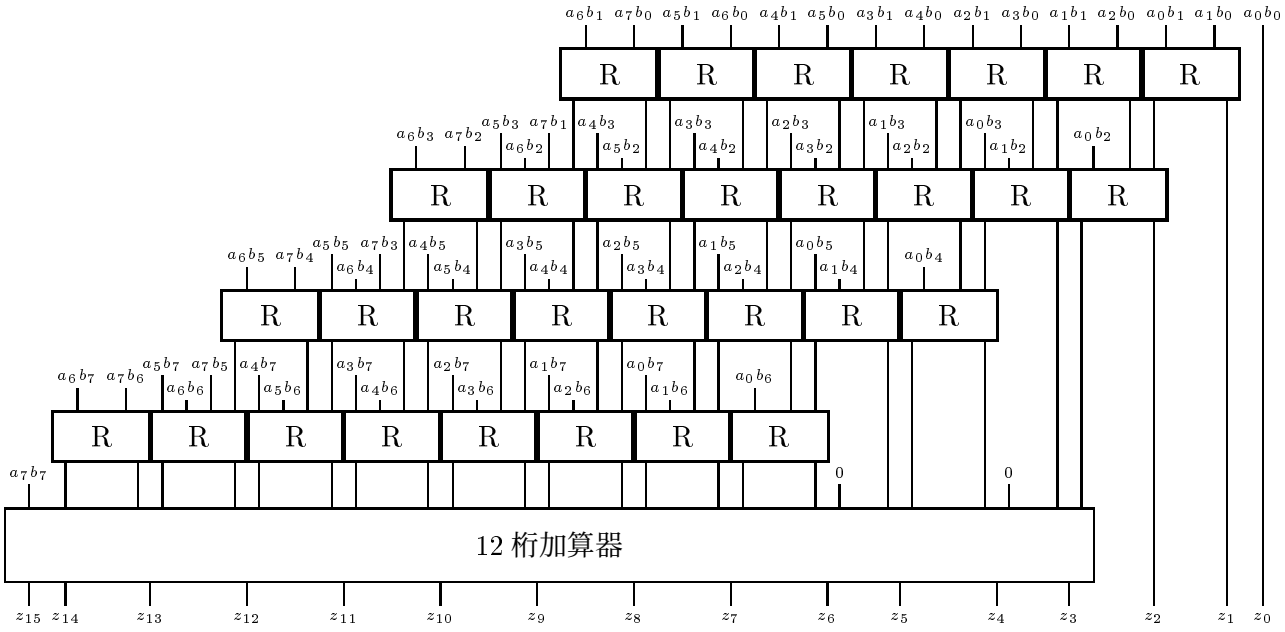


図 6.3: 対称 3 進表現での 8 桁乗算器

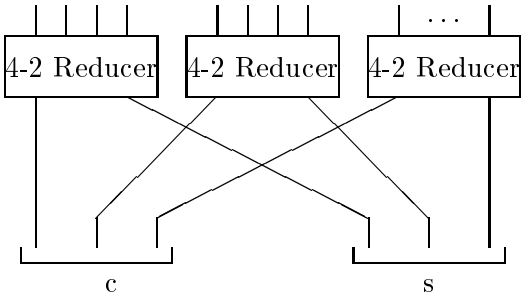
例えば 8 桁の乗算器は図 6.3 のようになる．偶数 n に対して，n 桁の対称 3 進表現での乗算器の CSA に必要な Reducer の個数は

$$\frac{n^2}{2} - 1 \tag{6.31}$$

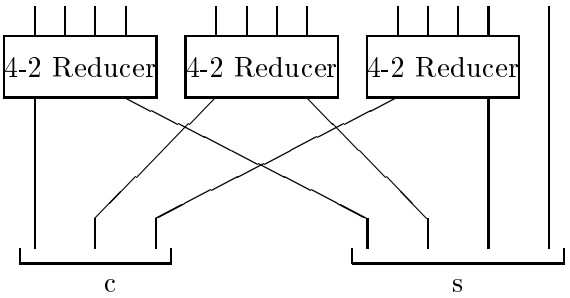
個である．

fa$n$ と同様に 4-2 Reducer を並べて，各 Reducer の c 同士と s 同士を集めたものを Rn と呼ぶことにする．

1. R10,R11,R12 の図



2. R13 の図



これらの図から，$n = 4k, 4k-1, 4k-2$ では，Rn の c と s は k 桁に，$n = 4k-3$ では c は k 桁で s は $k+1$ 桁になることが分かる．

6.4.4 3 値での配線の交差数

各 Rn の配線の交差数は fa$n$ と同様に c の桁数 c に依存し

$$1 + 2 + \cdots + c - 1 = \frac{c(c-1)}{2} \tag{6.32}$$

である．

対称 3 進表現での 6 桁の乗算を考える．2 進表現の場合と同様に部分積は

位	3^{10}	3^9	3^8	3^7	3^6	3^5	3^4	3^3	3^2	3^1	3^0
						•	•	•	•	•	•
					•	•	•	•	•	•	
				•	•	•	•	•	•		
			•	•	•	•	•	•			
		•	•	•	•	•	•				
	•	•	•	•	•	•					
個数	1	2	3	4	5	6	5	4	3	2	1

となる (ステップ 0)．各位の • を足すのに Rn を用いる．以下 2 進表現の場合と同様である．各ステップでの • の個数は次のようになる．

ステップ 0 の開始時		1	2	3	4	5	6	5	4	3	2	1
ステップ 1 の開始時		2	2	2	2	4	3	3	2	2	1	1
ステップ 2 の開始時	1	2	2	2	2	3	3	3	2	2	1	1
CPA への入力時	2	2	2	2	2	2	2	2	2	1	1	1

対称 3 進表現での k 桁 Wallace tree 乗算器の CSA 部において，ステップ m，3^i の位で使用する Rn の n を $n_{m,i}$ と書くとすると，次のアルゴリズム 6.3 によって $n_{m,i}$ により求めることができる．

アルゴリズム 6.3 Rn の n を求める

1. $m = 0$ のとき (初期値設定)

$$\begin{aligned} i < k \quad &\Rightarrow \quad n_{0,i} = i + 1, \\ i \geq k \quad &\Rightarrow \quad n_{0,i} = 2k - 1 - i, \end{aligned}$$

2. $m \geq 1$ のとき

$n_{m,i} =$ R$n_{m-1,i-1}$ の c の桁数 + R$n_{m-1,i}$ の s の桁数. □

§6.3.1 と同様に，各 Rn 間の配線の交差させないで配置配線させることができる．よって Wallace tree の配線の交差数は各 Rn 内の配線の交差数の合計となる．また，Wallace tree の配線数は Rn の n の合計となる．

6.5 2 進表現と対称 3 進表現での Wallace tree の比較

n ビット乗算器と対称 3 進表現での m 桁乗算器を考える．

$$m = n \cdot \log_3 2 \tag{6.33}$$

のとき，2 つの乗算器は同程度の整数を扱うことができる．(6.29) より n ビット乗算器の Wallace tree のノード数は $(n-1)^2$ であり，対称 3 進表現での m 桁乗算器のノード数は (6.31) より

$$\frac{m^2}{2} - 1 = \frac{(n \cdot \log_3 2)^2}{2} - 1 \approx 0.199n^2$$

である．よって，対称 3 進表現の乗算器の Wallace tree のノード数は，同規模の 2 進表現の乗算器の Wallace tree のノード数の約 1/5 になる．

表 6.1: 2 進表現と対称 3 進表現での Wallace tree の比較

2 進表現				対称 3 進表現				2 進/対称 3 進		
n	ノード数	配線数	配線の交差数	m	ノード数	配線数	配線の交差数	ノード数	配線数	配線の交差数
16	225	872	175	10	49	236	12	4.6	3.7	14.6
32	961	3,423	1,796	20	199	916	165	4.8	3.7	10.9
64	3,969	13,272	15,897	40	799	3,516	1,555	5.0	3.8	10.2
128	16,129	51,271	133,745	81	3,280	13,924	13,848	4.9	3.7	9.7
256	65,025	202,043	1,094,184	161	12,960	53,765	112,412	5.0	3.8	9.7
129	16,384	52,052	136,983	81	3,280	13,924	13,848	5.0	3.7	9.9

m は $2^n < 3^m$ となる最小の整数である．つまり対称 3 進表現のほうが規模の大きい乗算器である．$n = 128$ でも $n = 129$ でも $m = 81$ となるので，乗算器の規模を近づけるため，$n = 129$ も表記した．

n と m の関係 (6.33) とアルゴリズム 6.2 とアルゴリズム 6.3 及び (6.30), (6.32) から，対称 3 進表現を用いると Wallace tree の配線の交差数が約 1/10 になることが分かる．表 6.1 でいくつかのビット数と対応する 3 値論理の桁数に対して Wallace tree のノード数，配線数，配線の交差数差数をまとめた．

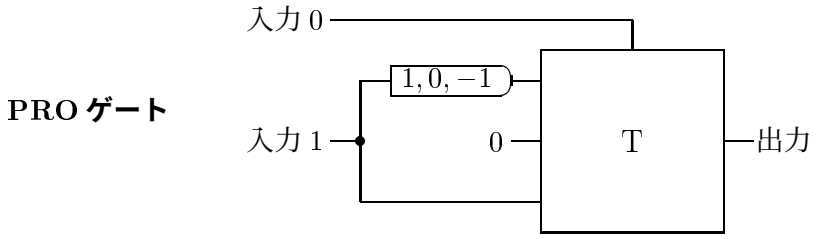
以上から対称 3 値論理を用いると，乗算器の Wallace tree のノード数は約 1/5，配線数は約 1/3.7，配線の交差数は約 1/10 となる．対称 3 値論理を用いると，ビット数 (桁数) を 2 倍にしてもまだ Wallace tree は単純になる．1 つの 4-2 Reducer は 3-2 Reducer より内部の配線は複雑で使用するトランジスタも多くなるが，Reducer はモジュールとしてあらかじめ構成することができるので，Reducer の乗算器の複雑さにはあまり影響しない．各 Reducer の配置と，Reducer 間の配線は広い範囲に影響するので，Reducer の個数の削減と Reducer 間の配線の簡単化が，乗算器の実装を容易にする．

6.6 乗算器の設計

§6.4 で対称 3 進表現での乗算器の概要を説明した．乗算器は 3 つのコンポーネント，MPS，CSA，CPA に分けられる．この節では各コンポーネントの設計法を提示する．

6.6.1 MPS

MPS は 1 桁の部分積を生成するコンポーネントであり，2 進表現では AND ゲートであるが，対称 3 進表現では PRO ゲートになる．



6.6.2 CSA

対称 3 進表現では 4-2 Reducer を Wallace tree に配置すると CSA になる．4-2 Reducer は本質的に

1 桁の値 + 1 桁の値 + 1 桁の値 + 1 桁の値

を行う加算器である．4-2 Reducer の入力を a, b, c, d，出力を $carry$ と s とすると，入力と出力の関係は表 6.2 のようになる．これは 2 進表現の 3-2 Reducer よりかなり複雑である．

a	b	c	d	carry	s	a	b	c	d	carry	s	a	b	c	d	carry	s
-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	0	1	-1	-1	-1	-1	1
-1	-1	-1	0	-1	0	0	-1	-1	0	-1	1	1	-1	-1	0	0	-1
-1	-1	-1	1	-1	1	0	-1	-1	1	0	-1	1	-1	-1	1	0	0
-1	-1	0	-1	-1	0	0	-1	0	-1	-1	1	1	-1	0	-1	0	-1
-1	-1	0	0	-1	1	0	-1	0	0	0	-1	1	-1	0	0	0	0
-1	-1	0	1	0	-1	0	-1	0	1	0	0	1	-1	0	1	0	1
-1	-1	1	-1	-1	1	0	-1	1	-1	0	-1	1	-1	1	-1	0	0
-1	-1	1	0	0	-1	0	-1	1	0	0	0	1	-1	1	0	0	1
-1	-1	1	1	0	0	0	-1	1	1	0	1	1	-1	1	1	1	-1
-1	0	-1	-1	-1	0	0	0	-1	-1	-1	1	1	0	-1	-1	0	-1
-1	0	-1	0	-1	1	0	0	-1	0	0	-1	1	0	-1	0	0	0
-1	0	-1	1	0	-1	0	0	-1	1	0	0	1	0	-1	1	0	1
-1	0	0	-1	-1	1	0	0	0	-1	0	-1	1	0	0	-1	0	0
-1	0	0	0	0	-1	0	0	0	0	0	0	1	0	0	0	0	1
-1	0	0	1	0	0	0	0	0	1	0	1	1	0	0	1	1	-1
-1	0	1	-1	0	-1	0	0	1	-1	0	0	1	0	1	-1	0	1
-1	0	1	0	0	0	0	0	1	0	0	1	1	0	1	0	1	-1
-1	0	1	1	0	1	0	0	1	1	1	-1	1	0	1	1	1	0
-1	1	-1	-1	-1	1	0	1	-1	-1	0	-1	1	1	-1	-1	0	0
-1	1	-1	0	0	-1	0	1	-1	0	0	0	1	1	-1	0	0	1
-1	1	-1	1	0	0	0	1	-1	1	0	1	1	1	-1	1	1	-1
-1	1	0	-1	0	-1	0	1	0	-1	0	0	1	1	0	-1	0	1
-1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	0	1	-1
-1	1	0	1	0	1	0	1	0	1	1	-1	1	1	0	1	1	0
-1	1	1	-1	0	0	0	1	1	-1	0	1	1	1	1	-1	1	-1
-1	1	1	0	0	1	0	1	1	0	1	-1	1	1	1	0	1	0
-1	1	1	1	1	-1	0	1	1	1	1	0	1	1	1	1	1	1

表 6.2: 4-2 Reducer の入出力の関係

6.6.3 4-2 Reducer の構成法

4-2 Reducer は 4 つの 1 桁の値の加算を行う演算器であることから，図 6.4 のようにして 4-2 Reducer を構成できることが分かる．

ところで，CAR の出力を反転させた関数 NCAR のゲートは CAR ゲートよりトランジスタ数が少なく性能もよい．それで ADD の出力を反転させた NADD ゲートと 1 桁の減算を行う SUB ゲートを導入すると，4-2 Reducer は図 6.5 のようになる．以上から，4-2 Reducer に必要な関数は，ADD，NADD，SUB，CAR，NCAR である．これらの関数のカルノー図は次のようになる．

ADD

	-1	0	1
-1	1	-1	0
0	-1	0	1
1	0	1	-1

NADD

	-1	0	1
-1	-1	1	0
0	1	0	-1
1	0	-1	1

SUB

	-1	0	1
-1	0	1	-1
0	-1	0	1
1	1	-1	0

CAR

	-1	0	1
-1	-1	0	0
0	0	0	0
1	0	0	1

NCAR

	-1	0	1
-1	1	0	0
0	0	0	0
1	0	0	-1

これらのゲートを TG 法で構成すると図 6.6 のようになり，トランジスタ数は

ゲート	ADD	NADD	SUB	CAR	NCAR
トランジスタ数	28	36	28	24	20

となる．

ところで，TG 法の長所は任意の 2 変数関数のゲートを簡単に構成できることと，遅延差が小さいことであり，欠点は必ず T ゲートを含むため，Olson 法よりトランジスタ数が多くなる場合があることである．できれば著者らの提案した TG 法だけで構成したいが，4-2 Reducer のように多く用いられるモジュールに対しては，Olson 法による 2 変数関数ゲートも考慮すべきである．各ゲートの Olson 法による構成は図 5.8, 6.7〜6.10 になり，トランジスタ数は

ゲート	ADD	NADD	SUB	CAR	NCAR
トランジスタ数	32	32	32	16	8

となる．トランジスタ数の観点から，CAR ゲートと NCAR ゲートは Olson 法により構成することが良い．また回路内のパスによってはトランスファーゲートを何回も通ることになり，そこを通る波形が悪くなる．それで，図 6.5 の ADD2 と NADD 及び NCAR は Olson 法を用いるとする．このとき 4-2 Reducer のトランジスタ数は 172 となる．2 進表現の 3-2 Reducer (FA) のトランジスタ数は 28 であるから，対称 3 進表現では Wallace tree のノード数が 1/5 になることを考慮しても，4-2 Reducer のトランジスタ数は 23%多くなる．しかしながら対称 3 進表現を用いる

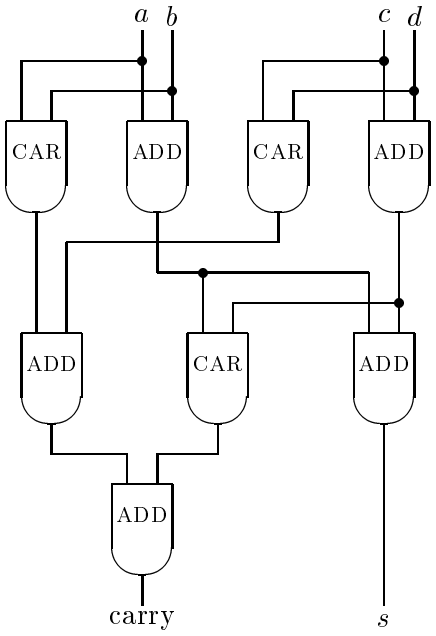


図 6.4: 4-2 Reducer

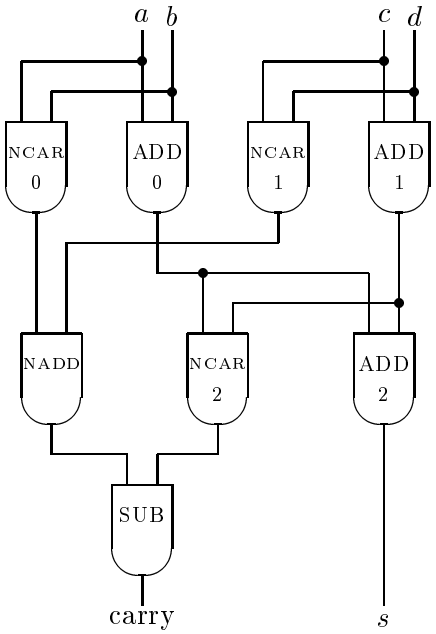


図 6.5: 改良された 4-2 Reducer

と，Wallace tree の配線は劇的に簡単なり配置配線が容易になるので，対称 3 進表現の乗算器は優れていると考えられる．トランジスタがより少なくなる 4-2 Reducer の構成法は今後の課題としたい．

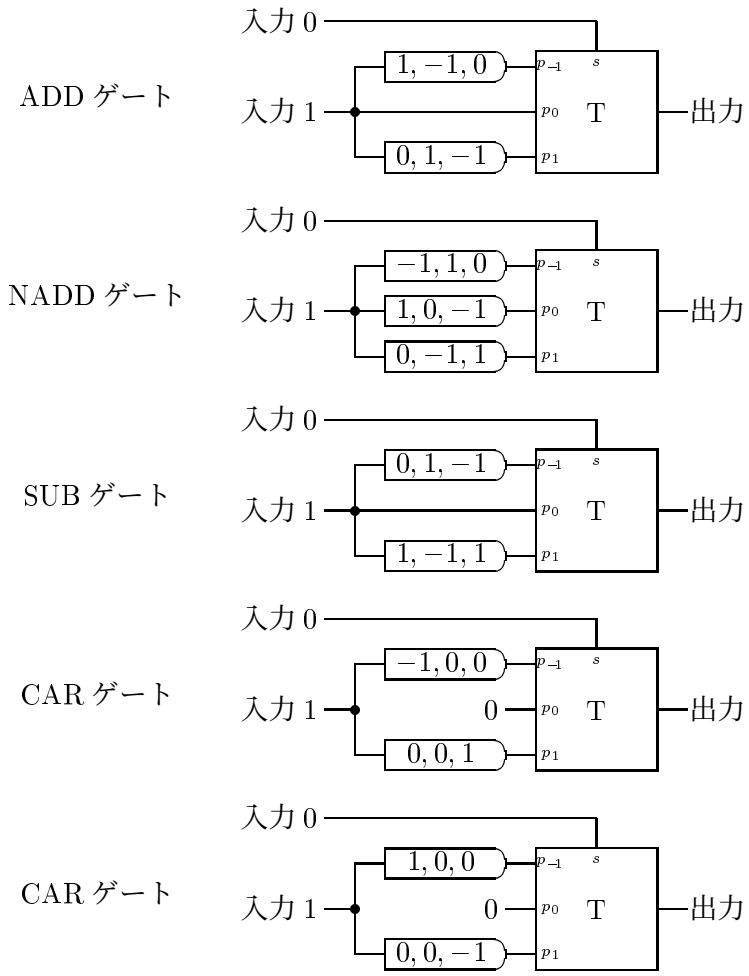


図 6.6: TG 法によるゲート構成

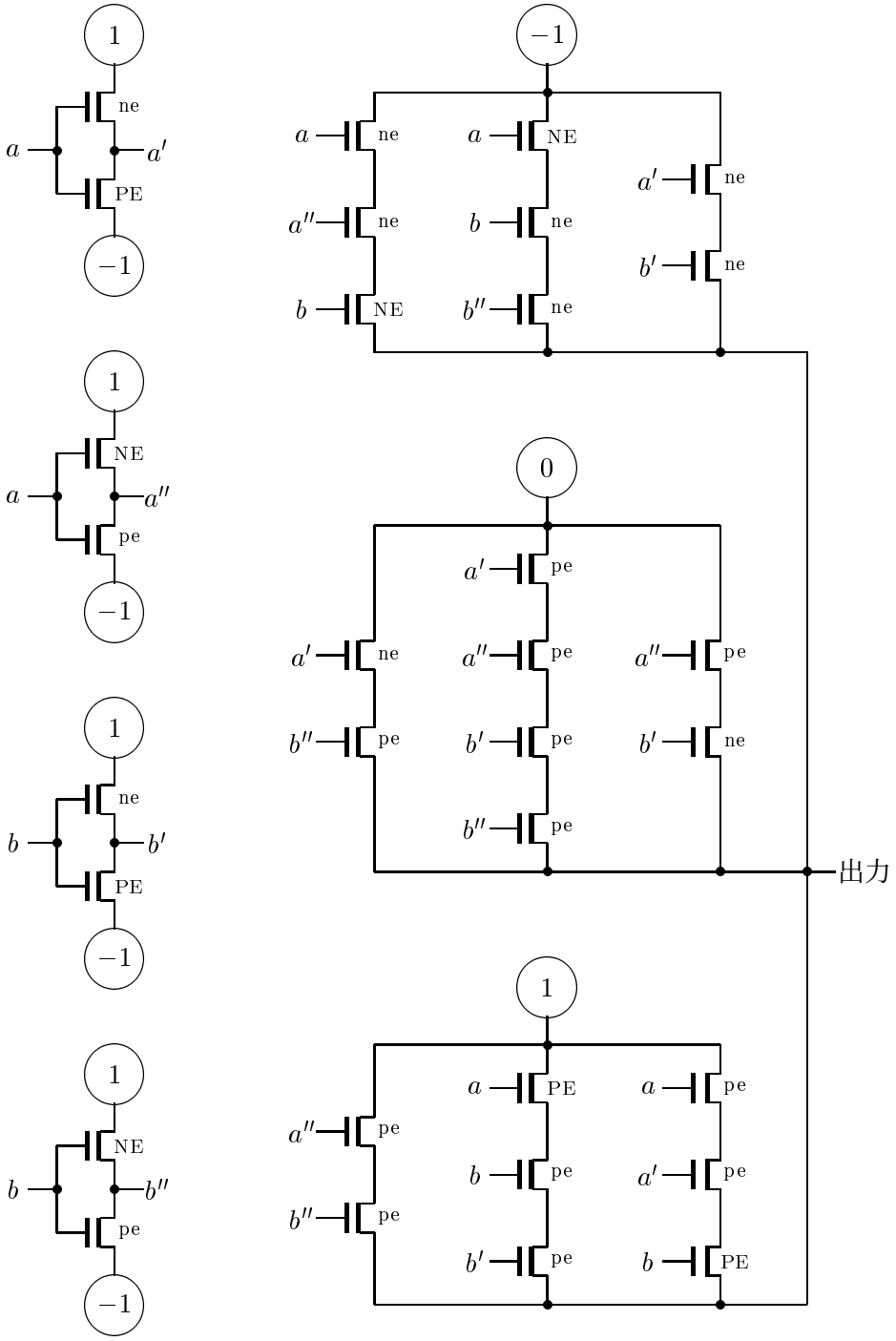


図 6.7: Olson 法による NADD ゲートの一例 (入力は a と b)

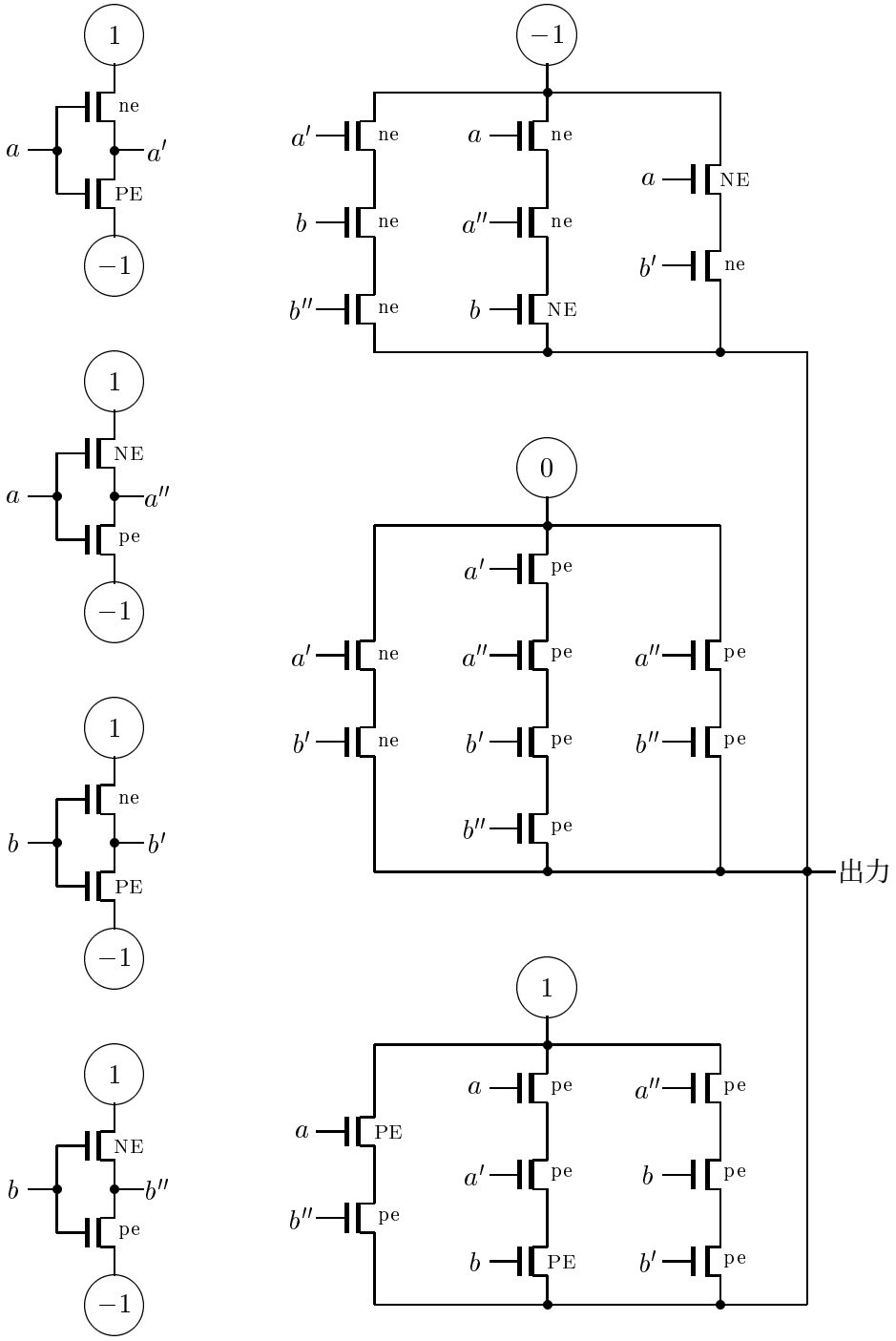


図 6.8: Olson 法による SUB ゲートの一例 (入力は a と b)

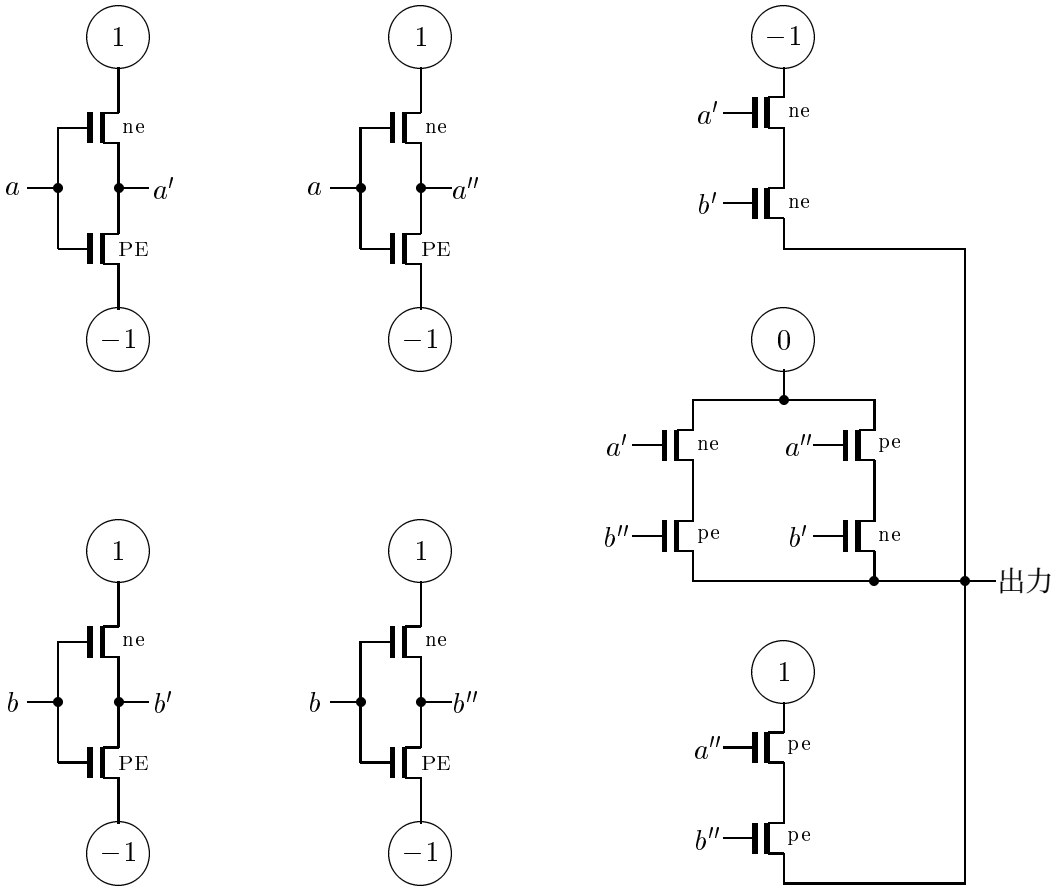


図 6.9: Olson 法による CAR ゲートの一例 (入力は a と b)

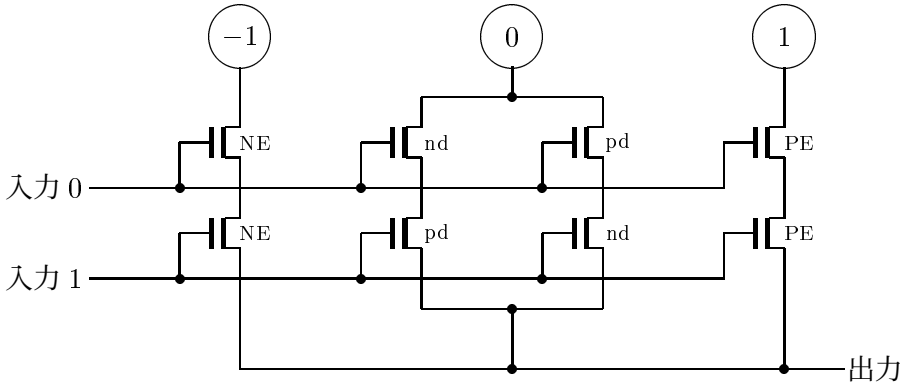


図 6.10: Olson 法による NCAR ゲートの一例

6.6.4 CPA

CPA は加算器と同じ構造であるので，加算器の構成を参照.

6.7 加算器の設計

ここでは，桁上げ先見加算のためのキャリー生成のために用いられるアルゴリズム 6.1 に必要なハードウェアを提示する．アルゴリズム 6.1 のステップ 1 に必要なゲートは CAR ゲートと ADD ゲートである．ステップ 2 の F_i は図 6.11 のような回路で生成できる．ステップ 3 に必要なゲートはステップ 1 と同じく CAR ゲートと ADD ゲートである．ステップ 4 の C'_i は図 6.12 のよう

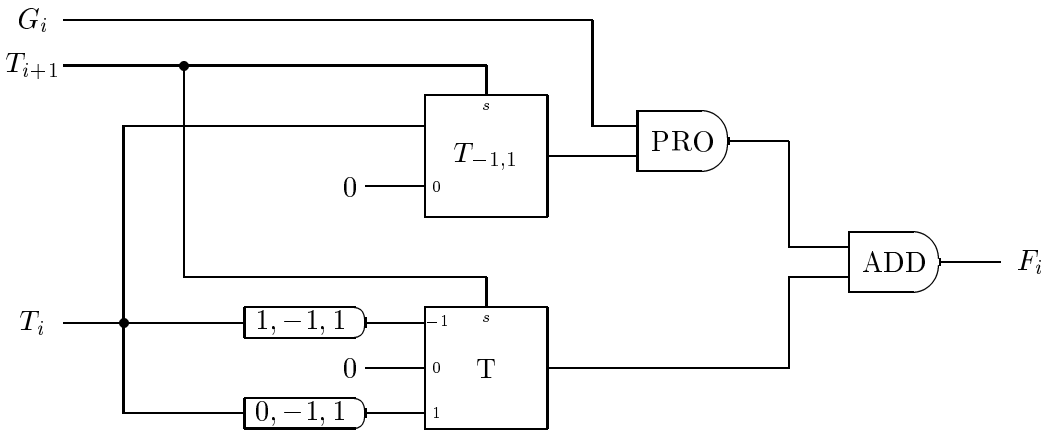


図 6.11: F_i 生成回路

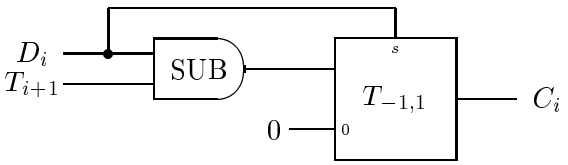


図 6.12: C'_i 生成回路

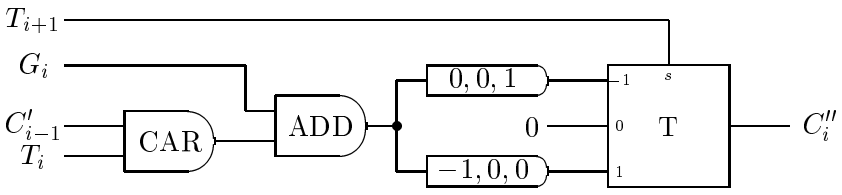


図 6.13: C''_i 生成回路

な回路で生成できる．ステップ 5 の C_i'' は図 6.13 のような回路で生成できる．ステップ 6 は 4-2 Reducer に A_i，B_i，C'_{i-1}，C''_{i-1} を入力すれば良い．

また，対称 3 進表現では，減算は片方の値を反転して加算を行えば良い．

6.8　6 章のまとめ

Wallace tree 乗算器は，論理段数がビット数の対数に比例する乗算器であるが，ビット数が多くなると Wallace tree の配線が非常に複雑になる．そのため，多ビットの Wallace tree 乗算器の実装は困難であった．本論文では，対称 3 進表現を用いると，扱える値が同規模の乗算器の Wallace tree のノード数が約 1/5，配線数が約 1/3.7，配線の交差数が約 1/10 になることを示した．表 6.1 より，対称 3 進表現を用いると $2n$ ビットに対応する桁数の対称 3 進表現での乗算器の Wallace tree が 2 進表現の n ビットのものより，簡単になることが分かる．但し，現時点では対称 3 進表現での Wallace tree のノード 1 つあたりのトランジスタ数が 2 進表現の約 6.3 倍となってしまい，乗算器のトランジスタ数は，対称 3 進表現を用いると 23%増加してしまう．しかし，集積回路ではトランジスタが占めている面積はチップの全面積の 30%程度であり，配線が占めている面積がトランジスタの占めている面積より 2 倍ほど大きいので，配線の複雑さの削減の効果は大きい．

RSA や標数が大きな有限体を用いる ElGamal 暗号では，暗号化/復号化に多ビットの乗算が必要になる．もし倍のビット数 (桁数) の乗算器を用意できれば，乗算器を使用する回数が 1/4 になり，暗号処理の効率化につながる．

第 7章

公開鍵暗号で用いられるアルゴリズムとそのハードウェア

この章では公開鍵暗号でよく用いられる演算を説明し，対称 3 進表現を用いてもそれらの類似を行うことができることを示す．またそのためのハードウェアについても提案する．

7.1 Montgomery 乗算と擬 Mersenne 素数

p を素数，$q = p^m$ とする．2 章で説明したように，RSA 暗号の暗号化/復号化では，公開鍵 N を法とする剰余乗算が必要である．有限体 $\mathbb{F}_p$ を用いる ElGamal 暗号でも，暗号化/復号化には p を法とする剰余乗算を数多く行う．また $q = p^n$ に対して $\mathbb{F}_q$ や楕円曲線を用いる ElGamal 暗号でも，p を法とする剰余乗算が必要である．

一般に乗算も剰余計算も時間のかかる演算である．暗号処理の効率性のためには，乗算と剰余計算を効率的に行う必要がある．

RSA 暗号では法とする値が公開鍵 N であるために，法とする値が一定でなく，そのような場合には Montgomery 乗算が用いられる．Montgomery 乗算については § 7.2 に記述されている．

ElGamal 暗号では p をシステムパラメータとするならば，法とする値が一定になる．そのため，剰余計算が効率的になる $p = 2^n - c,\ c$ は小さな整数という形の素数が選ばれる．このような形の素数を擬 Mersenne 素数という．擬 Mercennse 素数については § 7.3 に記述されている．

7.2 Montgomery 乗算

$R = 2^n$ とし (例えば，コンピュータのワード数に合わせて $n = 32$ など)，$p < R$ を奇数とする (素数でなくてもよい)．Montgomery 乗算では $x \in Z/pZ = \{0, 1, 2, \cdots, p-1\}$ (p は素数でないか

ら集合 $\{0, 1, 2, \cdots, p-1\}$ は一般には体ではないので，$\mathbb{F}_p$ とは書かない) の元を $xR \bmod p$ で表現する．以下の補題 7.1，アルゴリズム 7.1，7.2 はよく知られている結果である [8] が，後に対称 3 進表現での類似を考えるので詳しく説明する．

補題 7.1

$0 \leq y < pR$ に対して，$u = -yp^{-1} \bmod R$，$x = (y + up)/R$ とおくとき，x は $0 \leq x < 2p$ と $x \equiv yR^{-1} \pmod{p}$ を満たす整数である．

(証明)

奇数 p と $R = 2^n$ は互いに素だから，

$$\alpha p + \beta R = 1$$

を満たす整数 α, β が存在する．$\alpha p = 1 - \beta R$ であるから $\alpha p \equiv 1 \pmod{R}$ である．よって，$\alpha \equiv p^{-1} \pmod{R}$ である．

$$\begin{aligned} u = -yp^{-1} \bmod R &= -y\alpha \bmod R \\ y + up &\equiv y - y\alpha p \pmod{R} \\ &\equiv y - y(1 - \beta R) \pmod{R} \\ &\equiv y\beta R \pmod{R} \\ &\equiv 0 \pmod{R} \end{aligned}$$

よって $y + up$ は R で割切れるので，x は整数である．$0 \leq y < pR,\ 0 \leq up < pR$ なので，

$$x = (y + up)/R < (pR + pR)/R = 2p$$

となる．また，

$$x = (y + up)R^{-1} \equiv yR^{-1} \pmod{p}$$

である． □

ここで u の計算に $p^{-1}(= \alpha)$ を求めることが必要である．これはユークリッドの拡張アルゴリズムを使っても求められるが，効率的なアルゴリズムがある．

アルゴリズム 7.1

入力：奇数 x, $0 \leq x < 2^n$

出力：$y = x^{-1} \pmod{2^n}$

1. $y \leftarrow 1$
2. For i=2 to n do :
3. If $xy > 2^{i-1} \bmod 2^i$ then $y \leftarrow y + 2^{i-1}$
4. Return y

証明

数学的帰納法によってステップ 3 の終りで

$$xy \equiv 1 \pmod{2^i} \tag{7.1}$$

となることを示す．x は奇数だから $x = 2x' + 1$ と書ける．

(1) $i = 2$ のとき，$y = 1$ である．

1. $x' \equiv 0 \pmod 2$ のとき，$x' = 2B$ と書ける．
   $xy = x = 4B + 1 \equiv 1 \pmod{2^2}$ であり，ステップ 3 では何も行われないので $y = 1$ のままである．よって，(7.1) が成り立っている．
2. $x' \equiv 1 \pmod 2$ のとき，$x' = 2B + 1$ と書ける．
   $xy = x = 4B + 3 \equiv 3 \pmod{2^2}$ であり，ステップ 3 では $y \leftarrow y + 2^1 = 3$ となる．$xy = (B + 3) \cdot 3 = 12B + 9 \equiv 1 \pmod{2^2}$ となり，(7.1) が成り立っている．

(2) $xy \equiv 1 \pmod{2^{i-1}}$ と仮定する．このとき，$xy = A \cdot 2^{i-1} + 1$ と書ける．また x は奇数だから $x = 2x' + 1$ と書くことにする．

1. $A \equiv 0 \pmod 2$ のとき，$A = 2B$ と書ける．
   $xy = B \cdot 2^i + 1 \equiv 1 < 2^{i-1} \pmod{2^i}$ である．よってステップ 3 では何も行われず，その結果 $xy \equiv 1 \pmod{2^i}$ となる．よって (7.1) が成り立っている．
2. $A \equiv 1 \pmod 2$ のとき，$A = 2B + 1$ と書ける．
   $xy = B \cdot 2^i + 2^{i-1} + 1 \equiv 2^{i-1} + 1 > 2^{i-1} \pmod{2^i}$ である．$x(y + 2^{i-1}) = B \cdot 2^i + 2^{i-1} + 1 + (2x' + 1) \cdot 2^{i-1} = (B + x' + 1) \cdot 2^i + 1 \equiv 1 \pmod{2^i}$ となる．ステップ 3 では $y \leftarrow y + 2^{i-1}$ が行われるので，ステップ 3 の終りでは $xy \equiv 1 \pmod{2^i}$ となる．よって (7.1) が成り立っている． □

$x \in Z/pZ$ の元を $xR \bmod p$ とする表現では，加減算は $xR \bmod p$ と $yR \bmod p$ から $(x \pm y)R \bmod p$ を求めることになるが，これは普通に計算できる．乗算は

$$xR \bmod p \text{ と } yR \bmod p \text{ から } xyR \bmod p$$

を計算することが必要になる．まず普通に $(xR)(yR)$ を計算すると，$0 \leq (xR)(yR) = xyR^2 < p^2 < pR$ であるから，補題 7.1 を使って，$X \equiv xyR \pmod p$, $0 \leq X < 2p$ を計算できる．$X \geq p$ ならば $X \leftarrow X - p$ とすれば X が $xyR \pmod p$ である．この手法が Montgomery 乗算であり，mod R での乗算を効率的に行うことができる．

アルゴリズム 7.2 (Montgomery 乗算)
入力 $0 \leq X, Y < p$, 但し p は奇数
出力 $XYR^{-1} \pmod p$

1. $u \leftarrow -XYp^{-1} \pmod R$
2. $Z \leftarrow (y + up)/R$
3. If $Z \geq p$ then $Z \leftarrow Z - p$
4. Return Z □

ステップ 2 の R での割り算はシフトでできる．よって乗算結果 $\pmod p$ を求めるのに 3 回の乗算 $XY, (XY)p^{-1}, up$ でなされる．

7.3 擬 Mersenne 素数による剰余計算

$p = 2^n - k,\ k \leq \sqrt{2^n}$ という形の素数を擬 Mersenne 素数といい，このような p を法とする剰余計算アルゴリズムがある．

アルゴリズム 7.3
入力 n ビット整数 p と整数 x.
但し，$p = 2^n - k$，$0 \leq k < 2^{n/2}$，$0 \leq x \leq (p-1)^2$.
出力 $x \bmod p$.

(1) $x = q_0 \cdot 2^n + r_0$，$q_0 \geq 0$，$0 \leq r_0 < 2^n$ とする．
(q_0 は x の上位 n ビット，r_0 は x の下位 n ビット.)

$$x \ = \ \underbrace{\boxed{q_0}}_{n \text{ 桁}} \ \underbrace{\boxed{r_0}}_{n \text{ 桁}}$$

(2) $s_0 \leftarrow q_0 \cdot k + r_0$.

(3) $s_0 = q_1 \cdot 2^n + r_1$, $0 \le r_1 < 2^n$ とする.

(r_1 は s_0 の下位 n ビット,q_1 はそれより上のビット.)

$$s_0 \;=\; \boxed{q_1}\boxed{r_1}$$

(4) $s_1 \leftarrow q_1 k + r_1$, $s_2 \leftarrow (q_1+1)k + r_1$.

(5) $s_2 \ge 2^n$ ならば $(x \bmod p)$=$s_2 - 2^n$,

$s_2 < 2^n$ ならば $(x \bmod p)$=s_1.

$$s_2 \;=\; \boxed{1}\boxed{b_2} \quad \text{ならば } b_2 \text{ を出力}$$

$$s_2 \;=\; \boxed{0}\boxed{b_2} \quad \text{ならば } s_1 \text{ を出力}$$

□

s_0, s_1, s_2 の計算と s_2 のある 1 ビットが 0 か 1 かを判定することだけが, このアルゴリズムに必要な計算である.

7.4 対称 3 進表現での Montgomery 乗算

補題 7.1, アルゴリズム 7.1, アルゴリズム 7.2 の対称 3 進版を提案する. $R = 3^n$ とし, p は 3 と互いに素な自然数とする. また $0 < p < R$ とする.

補題 7.2 (補題 7.1 の対称 3 進版)
$-R^2/4 \le y \le R^2/4$ に対して, $u \equiv -yp^{-1} \pmod R$, $-(R-1)/2 \le u \le (R-1)/2$ とし $x = (y+up)/R$ とおくとき, x は $-3R/4 < x < 3R/4$ と $x \equiv yR^{-1} \pmod p$ を満たす整数である.

証明

p と $R = 3^n$ は互いに素だから,

$$\alpha p + \beta R = 1$$

を満たす整数 α, β が存在する. $\alpha p = 1 - \beta R$ であるから $\alpha p \equiv 1 \pmod R$ である. よって, $\alpha \equiv p^{-1} \pmod R$ である.

$$\begin{aligned} u = -yp^{-1} \bmod R &= -y\alpha \bmod R \\ y + up &\equiv y - y\alpha p \pmod R \\ &\equiv y - y(1-\beta R) \pmod R \\ &\equiv y\beta R \pmod R \\ &\equiv 0 \pmod R \end{aligned}$$

なので，$y+up$ は R で割切れるので，x は整数である．$-R^2/4 \le y < R^2/4,\ -R^2/2 < -R(R-1)/2 \le up < R(R-1)/2 \le R^2/2$ なので，

$$-\frac{3R}{4} < x = \frac{y+up}{R} < \frac{3R}{4}$$

となる．また，

$$x = (y+up)R^{-1} \equiv yR^{-1} (\mathrm{mod}\ p)$$

である． □

$p^{-1} \bmod R$ を求めるためにアルゴリズム 7.1 の類似があると良い．

アルゴリズム 7.4 (アルゴリズム 7.1 の対称 3 進版)

入力　：　3 と互いに素な整数 $x,\ -(3^n-1)/2 \le x \le (3^n-1)/2$

ここで x_0 を x の 3^0 の位とし $(x_0 = \pm 1)$，$x = 3x' + x_0$ とする．

出力　：　$y = x^{-1} \bmod 3^n,\ -(3^n-1)/2 \le y < (3^n-1)/2$

1. $y \leftarrow x_0$
2. For i=2 to n do :

   If xy の 3^{i-1} の位が 1 then $y \leftarrow y - x_0 \cdot 3^{i-1}$

   If xy の 3^{i-1} の位が -1 then $y \leftarrow y + x_0 \cdot 3^{i-1}$
3. Return y

証明

数学的帰納法によってステップ 3 の終りで

$$xy \equiv 1 \ (\mathrm{mod}\ 3^i) \tag{7.2}$$

となることを示す．$x_0 = \pm 1$ だから $x_0^2 = 1$ である．

(1) $i = 2$ のとき

$y = x_0$ であるから，$xy = (3x' + x_0)x_0 = 3x'x_0 + x_0^2$．よって $xy \equiv 1 \ (\mathrm{mod}\ 3)$ であり，$xy = 3A + 1$ と書ける．

1. $A \equiv 0 \ (\mathrm{mod}\ 3)$ のとき

   $A = 3B$ と書け，$xy = 9B$ とである．3 の位の値は 0 であるから，ステップ 2 では何も行われない．よって (7.2) が成り立っている．
2. $A \equiv 1 \ (\mathrm{mod}\ 3)$ のとき

$A = 3B + 1$ と書け，$xy = 9B + 3 + 1$ である．

$$\begin{aligned} x(y - x_0 \cdot 3) &= 9B + 3 - 3xx_0 + 1 \\ &= 9B + 3 - 9x'x_0 - 3x_0^2 + 1 \\ &= 9(B - x'x_0) + 1 \end{aligned}$$

となる．xy の 3 の位の値は 1 であるから，ステップ 2 では $y \leftarrow y - x_0 \cdot 3$ が行われるので，その結果 $xy \equiv 1 \pmod{3^2}$ となる．よって (7.2) が成り立っている．

3. $A \equiv 2 \pmod 3$ のとき

   $A = 3B - 1$ と書け，$xy = 9B - 3 + 1$ である．

$$\begin{aligned} x(y - x_0 \cdot 3) &= 9B - 3 + 3xx_0 + 1 \\ &= 9B - 3 + 9x'x_0 - 3x_0^2 + 1 \\ &= 9(B + x'x_0) + 1 \end{aligned}$$

   となる．xy の 3 の位の値は 1 であるから，ステップ 2 では $y \leftarrow y + x_0 \cdot 3$ が行われるので，その結果 $xy \equiv 1 \pmod{3^2}$ となる．よって (7.2) が成り立っている．

(2) $xy \equiv 1 \pmod{3^{i-1}}$ と仮定する．このとき，$xy = A \cdot 3^{i-1} + 1$ と書ける．

1. $A \equiv 0 \pmod 3$ のとき

   $A = 3B$ と書ける．$xy = B \cdot 3^i + 1$ であり 3^{i-1} の位は 0 であるからステップ 3 では何も行われない．その結果 $xy \equiv 1 \pmod{3^i}$ となる．よって (7.2) が成り立っている．

2. $A \equiv 1 \pmod 3$ のとき

   $A = 3B + 1$ と書ける．$xy = B \cdot 3^i + 3^{i-1} + 1$ であり，

$$\begin{aligned} x(y - x_0 \cdot 3^{i-1}) &= B \cdot 3^i + 3^{i-1} + 1 - (3x' + x_0)x_0 \cdot 3^{i-1} \\ &= B \cdot 3^i + 3^{i-1} + 1 - x'x_0 \cdot 3^i - x_0^2 \cdot 3^{i-1} \\ &= (B - x'x_0) \cdot 3^i + 1 \end{aligned}$$

   である．xy の 3^{i-1} の位は 1 なのでステップ 3 では $y \leftarrow y - x_0 \cdot 3^{i-1}$ が行われる．その結果 $xy \equiv 1 \pmod{3^i}$ となる．よって (7.2) が成り立っている．

3. $A \equiv 2 \pmod 3$ のとき

   $A = 3B - 1$ と書ける．$xy = B \cdot 3^i - 3^{i-1} + 1$ であり，

$$\begin{aligned} x(y + x_0 \cdot 3^{i-1}) &= B \cdot 3^i - 3^{i-1} + 1 + (3x' + x_0)x_0 \cdot 3^{i-1} \\ &= B \cdot 3^i - 3^{i-1} + 1 + x'x_0 \cdot 3^i + x_0^2 \cdot 3^{i-1} \\ &= (B + x'x_0) \cdot 3^i + 1 \end{aligned}$$

である．xy の 3^{i-1} の位は -1 なのでステップ 3 では $y \leftarrow y + x_0 \cdot 3^{i-1}$ が行われる．その結果 $xy \equiv 1 \pmod{3^i}$ となる．よって (7.2) が成り立っている． □

普通の Montgomery 乗算で使用する表現 ($x \in \mathbb{Z}/p\mathbb{Z}$ を $xR \bmod p$ と表現) と同様に，対称 3 進表現での Montgomery 乗算では $x \in \mathbb{Z}/p\mathbb{Z}$ を $xR \bmod p$, $-(R-1)/2 \leq xR \leq (R-1)/2$ に対応させる．このような表現の 2 元 $xR \bmod p$, $yR \bmod p$ に対して，加減算は $(x \pm y)R \bmod p$ を求めることであり，乗算は $xyR = (xR)(yR)R^{-1} \bmod p$ を求めることである．$p < R$ であるから $(xR)(yR)$ の範囲は

$$-\frac{R^2}{4} < -\frac{R-1}{2}\left(p - \frac{R-1}{2}\right) \leq (xR)(yR) \leq \frac{(R-1)^2}{4} < \frac{R^2}{4}$$

となる．よって $xyR = (xR)(yR)R^{-1} \pmod{p}$ を計算するのに補題 7.2 を用いることができる．

アルゴリズム 7.5 (対称 3 進での Montgomery 乗算)
入力 $-(R-1)/2 \leq X, Y \leq p - (R-1)/2$, 但し p は 3 の倍数でない
出力 $XYR^{-1} \pmod{p}$

1. $u \leftarrow -XYp^{-1} \pmod{R}$
2. $Z \leftarrow (y + up)/R$
3. If $Z > p - (R-1)/2$ then $Z \leftarrow Z - p$
   If $Z < p - (R-1)/2$ then $Z \leftarrow Z + p$
4. Return Z □

注意
普通の Montgomery 乗算では p は奇数，対称 3 進版では p は 3 と互いに素な整数でなければならないが，p として ElGamal 暗号では十分大きな素数，RSA では大きな素数の積を用いるので，p が 2 や 3 で割切れることはない．ElGamal 暗号では標数 2 や 3 の体を用いることがあるが，この場合は Montgomery 乗算を用いる必要はない．

7.5 $3^n - k$ という形の素数を法とする剰余計算アルゴリズム

対称 3 進表現では $3^n - k$, k は小さな整数，という形の素数が擬 Mersenne 素数に対応する．p が $p = 3^n - k$, $k^2 + k - 1 < 3^n$ という形の場合，§7.3 と同様なアルゴリズムがあることを示す．一般に $x \bmod p$ は

$$\begin{cases} y \equiv x \pmod{p} \\ 0 \leq y \leq p-1 \end{cases}$$

を満たす整数 y のことだが，ここでは対称 3 進表現を考えているので，

$$\begin{cases} y' \equiv x \pmod{p} \\ -\dfrac{3^n-1}{2} \le y' \le \dfrac{3^n-1}{2} - k \end{cases}$$

を満たす整数 x で $\mathbb{F}_p$ の元を表す必要がある．このような y' を

$$x \text{ Mod } p$$

と表記することにする．

この節では

$$-(3^n-1)(3^n-1-k) \le x \le (3^n-1)^2$$

という範囲にある x に対して，x Mod p を求めるための剰余計算アルゴリズムを考える．$-(3^n-1)/2 \le a, b, c, d \le (3^n-1)/2 - k$ に対して，

$$-(3^n-1)(3^n-1-k) \le (a \pm b)(c \pm d) \le (3^n-1)^2$$

となるので，今回考える剰余計算アルゴリズムは

$$(a \pm b)(c \pm d) \text{ Mod } p \tag{7.3}$$

を求めるのに適している．(7.3) という計算は Montgomery 型楕円曲線による暗号で数多くなされる演算である [28, 29].

アルゴリズム 7.6 ($3^n - k$ という形の素数を法とする剰余計算アルゴリズム)

入力 対称 3 進表現の n 桁整数 p と整数 x.
但し，$p = 3^n - k$，$k^2 + k - 1 < 3^n$，$-(3^n-1)(3^n-1-k) \le x \le (3^n-1)^2$.

出力 x Mod p.

(1) $x = q_0 \cdot 3^n + r_0$，$-(3^n-1)/2 \le r_0 \le (3^n-1)/2$ とする.
(q_0 は x の上位 $n+1$ 桁，r_0 は x の下位 n 桁.)

$$x \;=\; \underbrace{\boxed{q_0}}_{n+1\text{桁}}\underbrace{\boxed{r_0}}_{n\text{桁}}$$

(2) $s_0 \leftarrow q_0 \cdot k + r_0$.

(3) $s_0 = q_1 \cdot 3^n + r_1$，$-(3^n-1)/2 \le r_1 \le (3^n-1)/2$ とする.
(r_1 は s_0 の下位 n 桁,q_1 はそれより上の桁.)

$$s_0 \;=\; \boxed{q_1}\boxed{r_1}$$

(4) $s_1 \leftarrow q_1 k + r_1$,

$s_2 \leftarrow (q_1 + 1)k + r_1$.

$s_3 \leftarrow (q_1 - 1)k + r_1$.

(5) $s_1 = h_1 \cdot 3^n + b_1$,

$s_2 = h_2 \cdot 3^n + b_2$,

$s_3 = h_3 \cdot 3^n + b_3$,

$$s_1 = \boxed{h_1}\boxed{b_1}$$

$$s_2 = \boxed{h_2}\boxed{b_2}$$

$$s_3 = \boxed{h_3}\boxed{b_3}$$

(6) $h_2 = 1$ ならば, x Mod p=b_2,

$h_2 \neq 1, h_1 = -1$ ならば, x Mod p=b_3,

$h_2 \neq 1, h_1 \neq -1$ ならば, x Mod p=b_1. □

このアルゴリズムで必要な演算は，s_0, s_1, s_2, s_3 を求めることとステップ 6 での条件判定だけである．

アルゴリズム 7.6 を証明する前に，いくつかの補題を議論する．

補題 7.3

n+1 桁の対称 3 進表現で表現された整数 y に対して

$$\begin{aligned}
\text{最上位の桁が } 1 &\Leftrightarrow y \geq \frac{3^n + 1}{2} \\
\text{最上位の桁が } 0 &\Leftrightarrow -\frac{3^n - 1}{2} \leq y \leq \frac{3^n - 1}{2} \\
\text{最上位の桁が } -1 &\Leftrightarrow y \leq -\frac{3^n + 1}{2}
\end{aligned}$$

証明

最上位が 1 の最小の値は

$$\begin{aligned}
(1, -1, \cdots, -1, -1)_3 &= 3^n - 3^{n-1} - \cdots - 3 - 1 \\
&= 3^n - (1 + 3 + \cdots + 3^{n-1}) \\
&= 3^n - \frac{3^n - 1}{3 - 1} \\
&= 3^n - \frac{3^n - 1}{2} \\
&= \frac{3^n + 1}{2}
\end{aligned}$$

同様に最上位が -1 のときの最大の値は

$$(-1, 1, \cdots, 1)_3 = \frac{-3^n - 1}{2}$$

である. □

次の 2 つの補題はアルゴリズムに出てくる q_0 の範囲を求める.

補題 7.4

$(3^n - 1)^2$ を対称 3 進表現で表すと

$$\begin{array}{cccccccc|ccccc} (& 1 & 0 & \cdots & 0 & -1 & 1 & & 0 & \cdots & 0 & 1 &)_3 \\ \text{桁} & 2n & 2n-1 & & n+2 & n+1 & n & & n-1 & & 1 & 0 & \end{array} \tag{7.4}$$

証明

$$\begin{aligned} (3^n - 1)^2 &= 3^{2n} - 2 \cdot 3^n + 1 \\ &= 3^{2n} + (-3 + 1) \cdot 3^n + 1 \\ &= 3^{2n} - 3^{n+1} + 3^n + 1 \end{aligned}$$

よって $(3^n - 1)^2$ の対称 3 進表現は (7.4) のようになる. □

補題 7.5

アルゴリズム 7.6 の q_0 に対して $-(3^n - 2) \leq q_0 \leq 3^n - 2$ となる.

証明

q_0 が最大となるのは $x = (3^n - 1)^2$ のときであり, 補題 7.4 より $q_0 = (1, 0, \cdots, 0, -1, 1)_3 = 3^n - 2$ である. 同様に q_0 の最小値は $-(3^n - 2)$ 以上である. □

これで, アルゴリズム 7.6 の証明の準備ができた.

アルゴリズム 7.6 の証明

アルゴリズムの k に関する条件から,

$$\begin{aligned} k^2 + k - 1 &< 3^n \\ \frac{k^2 + k - 1}{3^n} &< 1 \end{aligned} \tag{7.5}$$

となる. 更に

$$\begin{aligned} k^2 &< 3^n \\ 3^n + k^2 - 2k &< 2 \cdot 3^n - 2k \\ \frac{3^n + k^2 - 2k}{3^n - k} &< 2 \end{aligned} \tag{7.6}$$

となる．

n 桁の対称 3 進表現の整数の値の範囲は $-(3^n-1)/2$ 以上 $-(3^n-1)/2$ 以下である．また，アルゴリズムの操作から

$$x = q_0 \cdot 3^n + r_0,\ -\frac{3^n-1}{2} \le r_0 \le \frac{3^n-1}{2} \tag{7.7}$$

$$s_0 = q_0 \cdot k + r_0 = q_1 \cdot 3^n + r_1,\ -\frac{3^n-1}{2} \le r_1 \le \frac{3^n-1}{2} \tag{7.8}$$

$$s_1 = q_1 k + r_1,\ s_2 = q_1 k + r_1 + k,\ s_3 = q_1 k + r_1 - k \tag{7.9}$$

という関係が成り立っている．

補題 7.5 より，

$$-(3^n-2) \le q_0 \le 3^n - 2$$

である．$R = x \bmod p$ とおくと，$p = 3^n - k$ であるから，

$$x = (q_0 + \mathcal{Q})(3^n - k) + \mathcal{R},\ -\frac{3^n-1}{2} \le \mathcal{R} \le \frac{3^n-1}{2} - k \tag{7.10}$$

と書ける．つまり，$(q_0 + \mathcal{Q})$ は商 x/p の整数部である．以下では

(I) $\mathcal{Q}$ は q_1, q_2, q_3 のどれかに等しくなること
(II) アルゴリズム 7.6 の出力が $\mathcal{R}$ に等しくなること

を示す．

(7.7) と (7.10) から

$$\begin{aligned} q_0 \cdot 3^n + r_0 &= (q_0 + \mathcal{Q})(3^n - k) + \mathcal{R} \\ &= q_0 \cdot 3^n - q_0 k + \mathcal{Q} \cdot 3^n - \mathcal{Q}k + \mathcal{R} \\ r_0 - \mathcal{R} &= -q_0 k + \mathcal{Q} \cdot 3^n - \mathcal{Q}k \end{aligned} \tag{7.11}$$

$-\frac{3^n-1}{2} \le r_0 \le \frac{3^n-1}{2},\ -\frac{3^n-1}{2} \le \mathcal{R} \le \frac{3^n-1}{2} - k$ であるから，

$$-(3^n-1) < -(3^n-1) + k \le r_0 - \mathcal{R} \le 3^n - 1 \tag{7.12}$$

である．

(7.11) と (7.12) の右の不等式から $\mathcal{Q}$ の上限を求めることができる．

$$\begin{aligned} r_0 - \mathcal{R} = -q_0 k + \mathcal{Q} \cdot 3^n - \mathcal{Q}k &< 3^n && \text{(7.11), (7.12) より} \\ \mathcal{Q}(3^n - k) &< 3^n + q_0 k \le 3^n + (3^n - 2)k && \text{補題 7.5 より} \\ \mathcal{Q} &< \frac{3^n + (3^n-2)k}{3^n - k} \\ &= k + \frac{3^n + k^2 - 2k}{3^n - k} \\ &< k + 2 && \text{(7.6) より} \end{aligned}$$

$\mathcal{Q}$ は整数だから $\mathcal{Q} \leq k+1$ であることが分かる．同様に $-(k+1) \leq \mathcal{Q}$ である．よって

$$-(k+1) \leq \mathcal{Q} \leq k+1 \tag{7.13}$$

である．

(7.8) と (7.10) より

$$\begin{aligned} \mathcal{R} &= r_0 + q_0 k - \mathcal{Q} \cdot 3^n + \mathcal{Q}k \\ &= r_1 + q_1 \cdot 3^n - \mathcal{Q} \cdot 3^n + \mathcal{Q}k \\ &= (q_1 - \mathcal{Q}) \cdot 3^n + r_1 + \mathcal{Q}k \qquad (7.14) \\ (\mathcal{Q} - q_1) \cdot 3^n &= r_1 + \mathcal{Q}k - \mathcal{R} \\ \mathcal{Q} - q_1 &= \frac{r_1 + \mathcal{Q}k - \mathcal{R}}{3^n} \end{aligned}$$

となる．$-(3^n-1)/2 \leq r_1 \leq (3^n-1)/2,\ (3^n-1)/2 + k \leq -\mathcal{R} \leq (3^n-1)/2$ であり，更に (7.13) より $-(k+1) \leq \mathcal{Q} \leq k+1$ なので，

$$\mathcal{Q} - q_1 \leq \frac{3^n - 1 + (k+1)k}{3^n} = 1 + \frac{k^2+k-1}{3^n}$$

$$-1 - \frac{k^2+k-1}{3^n} < \frac{-(3^n-1)+k-(k+1)k}{3^n} \leq \mathcal{Q} - q_1$$

となる．(7.5) より $-2 < \mathcal{Q} - q_1 < 2$ である．$\mathcal{Q} - q_1$ は整数だから，$-1 \leq \mathcal{Q} - q_1 \leq 1$ である．よって (I) が示された．

次に (II) を示す．(7.9) より，s_1, s_2, s_3 の関係

$$s_1 = s_2 - k = s_3 + k$$

$$s_2 = s_1 + k = s_3 + 2k$$

$$s_3 = s_1 - k = s_2 - 2k$$

が得られる．これ以降では関係式 (7.14) と補題 7.3 を用いる．各 i と $n+1$ 桁の s_i に対して，h_i は s_i の上位 1 桁，b_i は s_i の下位 n 桁を表している．

1. $\mathcal{Q} = q_1$ のとき

   (7.14) より $\mathcal{R} = r_1 + \mathcal{Q}k = r_1 + q_1 k = s_1$.

$$-\frac{3^n-1}{2} \leq s_1 = \mathcal{R} \leq \frac{3^n-1}{2} - k,\ \text{よって } h_1 = 0$$

$$-\frac{3^n-1}{2} + k \leq s_2 = s_1 + k = \mathcal{R} + k \leq \frac{3^n-1}{2},\ \text{よって } h_2 = 0$$

$$-\frac{3^n-1}{2} - k \leq s_3 = s_1 - k = \mathcal{R} - k \leq \frac{3^n-1}{2} - 2k,\ \text{よって } h_3 = -1 \text{ または } 0$$

   このとき，アルゴリズム 7.6 は b_1 を出力するが，$b_1 = s_1 = \mathcal{R}$ である．

2. $\mathcal{Q} = q_1 + 1$ のとき

(7.14) より $\mathcal{R} = r_1 + \mathcal{Q}k - 3^n = r_1 + (q_1 + 1)k - 3^n = s_2 - 3^n$.

$$\frac{3^n + 1}{2} - k \leq s_1 = s_2 - k = \mathcal{R} + 3^n - k \leq \frac{3^{n+1} - 1}{2} - 2k, \text{ よって } h_1 = 0 \text{ または } 1$$

$$\frac{3^n + 1}{2} \leq s_2 = \mathcal{R} + 3^n \leq \frac{3^{n+1} - 1}{2} - k, \text{ よって } h_2 = 1$$

$$\frac{3^n + 1}{2} - 2k \leq s_3 = s_2 - 2k = \mathcal{R} + 3^n - 2k \leq \frac{3^{n+1} - 1}{2} - 3k, \text{ よって } h_3 = 0 \text{ または } 1$$

このとき，アルゴリズム 7.6 は b_2 を出力するが，$b_2 = s_2 - 3^n = \mathcal{R}$ である．

3. $\mathcal{Q} = q_1 - 1$ のとき

(7.14) より $\mathcal{R} = r_1 + \mathcal{Q}k + 3^n = r_1 + (q_1 - 1)k + 3^n = s_3 + 3^n$.

$$-\frac{3^{n+1} - 1}{2} + k \leq s_1 = s_3 + k = \mathcal{R} - 3^n + k \leq -\frac{3^n + 1}{2}, \text{ よって } h_1 = -1$$

$$-\frac{3^{n+1} - 1}{2} + 2k \leq s_2 = s_3 + 2k = \mathcal{R} - 3^n + 2k \leq -\frac{3^n + 1}{2} + k, \text{ よって } h_2 = -1 \text{ または } 0$$

$$-\frac{3^{n+1} - 1}{2} \leq s_3 = \mathcal{R} - 3^n \leq -\frac{3^n + 1}{2} - k, \text{ よって } h_3 = -1$$

このとき，アルゴリズム 7.6 は b_3 を出力するが，$b_3 = s_3 + 3^n = \mathcal{R}$ である．

以上で証明終り． □

7.6 Montgomery 乗算に必要な演算器

Montgomery 乗算は加減乗算器だけで行うことができるが，§7.4 のアルゴリズム 7.4 のステップ 2 を行うためのハードウェアがあると，効率的に行うことができる．n 桁の値を考える場合，このためのハードウェアの概要は図 7.1 のようになる．カウンタは 1 から開始し n までクロックごとに 1 つ値を増やす．カウンタの値はシフタ，セレクタ，分配器の制御信号となる．シフタは制御信号の値だけ左シフトを行う．加減算器は，$2n$ 桁の入力で，制御信号が 1 のときは加算を行い，制御信号が -1 のときは減算を行い，制御信号が 0 のときは左の入力をそのまま出力する．セレクタは n 桁の入力から，制御信号桁目を出力する．分配器はセレクタと逆で，入力は 1 桁，出力は n 桁で，入力の値を出力の制御信号桁目に送り，その他の桁には 0 をセットする．

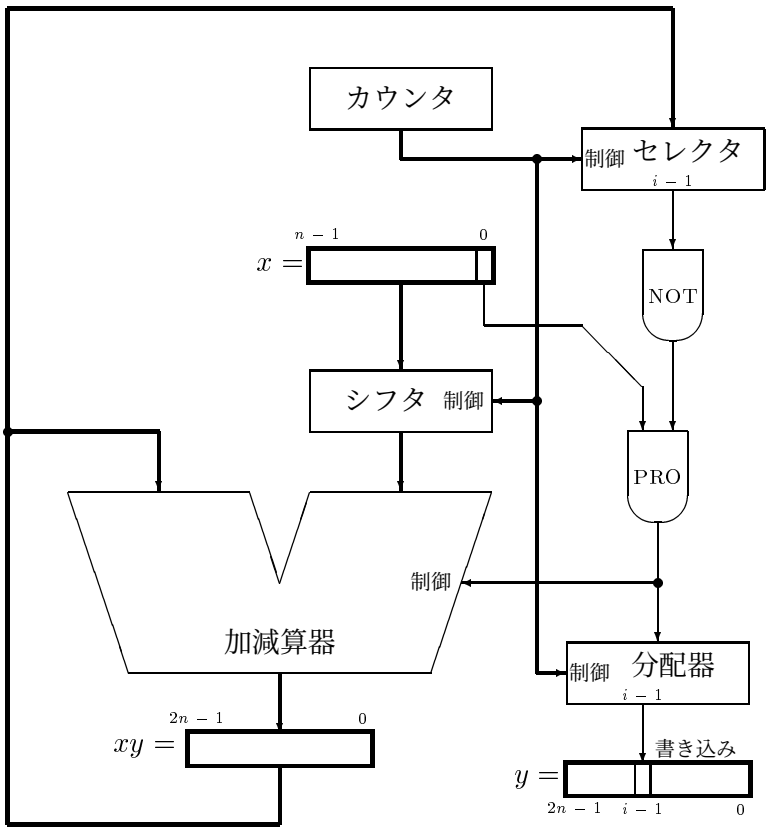


図 7.1: Montgomery 乗算のためのハードウェアの概要

7.7 $3^n - k$ という形の素数を法とする剰余計算に必要な演算器

§7.5 のアルゴリズム 7.6 のステップ (5) 以外は乗算器と加算器だけで良いが，ステップ (5) には特別な回路が必要である．この回路には縮退 T ゲートの $T_{0,1}$ ゲート (§ 4.3.6) を用いると良い．すると，アルゴリズム 7.6 のステップ (5) のための回路は図 7.2 のようになる．

制御信号	出力
-1	a
0	b
1	b

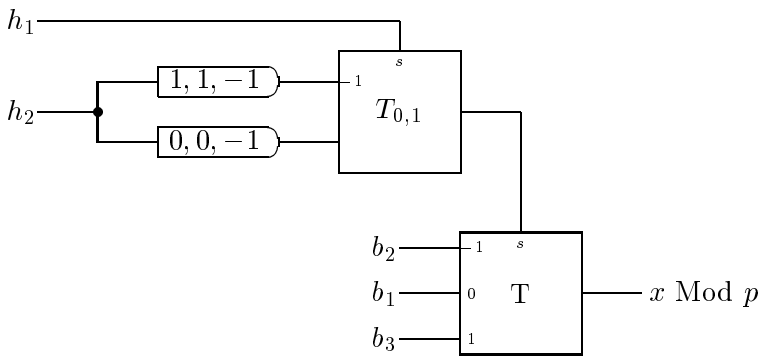


図 7.2: アルゴリズム 3.8 ステップ (5) のための回路

7.8 7 章のまとめ

6 章では，対称 3 進表現を用いると，Wallace tree 乗算器の配線が容易になるため，ビット数 (桁数) が 2 倍の乗算器が実装できることを示唆した．しかし，乗算以外の公開鍵暗号に必要な演算やアルゴリズムも対称 3 進表現で行うことができなければ，公開鍵暗号に対称 3 進表現を用いることができない．

RSA 暗号も ElGamal 暗号も暗号化/復号化に剰余計算が必要であるが，剰余計算を効率的に行う方法として，Montgomery 乗算と擬 Mersenne 素数を法とする剰余計算がよく用いられている．これらは従来の計算機が用いている 2 進表現に特化している．7 章では，これらの類似は，対称 3 進表現でも可能であることを示した．Montgomery 乗算と擬 Mersenne 素数の対称 3 進版は本研究の成果である．よって公開鍵暗号の処理のすべてを対称 3 進表現で行うことができることが示された．

第 8章

結言

第 2 章では，よく用いられている公開鍵暗号の RSA 暗号と ElGamal 暗号の暗号化/ 復号化の方法を説明し，また ElGamal 暗号で必要となる有限体について解説した．有限体については，特に計算の効率性を得るための ONB と呼ばれる基底について詳しく解説した．

第 3 章では，ElGamal 暗号の一種である XTR について解説した．XTR は同じ安全性に対して，公開鍵のサイズを ElGamal 暗号の 1/3 にでき，暗号化/復号化処理も効率的であるため，近年注目されている．本論文では，XTR に標数 3 の体を用いると更に公開鍵のサイズを半分にできることを示した．また楕円曲線暗号にも標数 3 の体を使うことが提案されている．それで，標数 3 の体の実装も興味深い対象となっている．なお，楕円曲線暗号も ElGamal 暗号の一種である．

第 4 章では，3 値論理ゲートの構成法として，Olson 法を解説し，3 値論理ゲートの組織的な構成法として TG 法を提案した．TG 法と Olson 法によるデバイスは，CMOS デバイスと同様に定常時には電流が流れないことと，現在の CMOS デバイスの加工技術で製造できるという点で，3 値論理ゲートのための優れた構成法である．TG 法は Olson 法によって構成した 1 変数関数ゲートと T ゲートをモジュールとして，任意の関数のゲートを構成する方法である．すべて Olson 法を用いる方法より，性能や任意の関数のゲートの構成の容易さから，TG 法が優れている．また，HSPICE のシミュレーションに用いる MOSFET の閾値のためのパラメータも決定した．

標数 3 の体の実装は 2 値論理では無駄が多くなるが，3 値論理では効率的に実装できる．このことが 3 値論理の研究の最初の動機であった．

第 5 章では，第 4 章の 3 値論理ゲートを用いて，標数 3 の体での XTR のためのハードウェアを設計し，HSPICE シミュレーションにより性能を評価した．この結果，現時点での TG 法による 3 値論理ゲートの性能は十分ではないが，今後の微細加工技術により性能向上が期待できる．この性能向上の度合いは 2 値論理ゲートより大きいことが期待できる．

第 6 章では，加算器と乗算器に対称 3 進表現を用いることを提案した．性能を追求すると，加算器には桁上げ先見加算器，乗算器は Wallace tree 乗算器が必要である．桁上げ先見器にはキャ

リー生成アルゴリズムが必要であるが，対称 3 進表現でも同様のアルゴリズムがあることを示し，その正当性を証明した．Wallace tree 乗算器は高性能であるが，配線が非常に複雑になるために，多ビットの Wallace tree 乗算器の実装が困難であった．ところが対称 3 進表現で Wallace tree 乗算器を構成すると，Wallace tree のノード数は約 1/5 に，配線数は約 1/3.7 に，配線の交差数は 1/10 となり，Wallace tree の構造が非常に簡単になる．つまり対称 3 進表現を用いると，よりビット数 (桁数) の大きい乗算器を実装できる．多くの公開鍵暗号は非常にビット数の大きな値の乗算を必要とするが，多ビット乗算器は乗算器の使用回数を削減する．例えば，128 ビット乗算を行うのに，32 ビット乗算器は 16 回使用しなければならないが，64 ビット乗算器では 4 回の使用で済む．このことが，多ビット乗算を必要とする多くの公開鍵暗号に対称 3 進表現が適している点である．

第 7 章では，公開鍵暗号でよく用いられるアルゴリズムである「Montgomery 乗算」と「擬 Mersenne 素数を法とする剰余計算」を解説し，これらのアルゴリズムの対称 3 進版を与え，アルゴリズムの正当性を証明した．元来のこれらのアルゴリズムは 2 進表現に特化したものであるが，本論文で提案した対称 3 進版もほとんど同じ手順で行うことができる．また，これらに必要なハードウェアについても提示した．

最後に公開鍵暗号に必要な演算とアルゴリズムをまとめておく．

	演算	アルゴリズム	3 値論理の適否
ElGamal 暗号 (小さな標数 p)	p を法とする加算と乗算	–	$p = 3$ ならば適
ElGamal 暗号 (大きな標数 p)	$\|p\|$ ビットの加算と乗算	擬 Mersenne 素数剰余計算	適
RSA 暗号 (公開鍵 N)	$\|N\|$ ビットの乗算	Montgomery 乗算	適

よってほとんどの公開鍵暗号に 3 値論理が適している．

本研究の成果が，多値論理の研究の活性化の，例えば，画期的な多値論理ゲートの開発や多値論理に実用化等の，動機付けとなることを期待する．

第 A 章

HSPICEのパラメータ

本研究は [25] が公開していた $0.1\mu m$ プロセスのための HSPICE パラメータを用いてシミュレーションを行ったが，現在はデータが更新され，$0.1\mu m$ プロセスのパラメータはアクセスできなくなった．そこで，ここに本研究が使ったパラメータを記す．なお，V_{th0} と Nch は §6.3 で求めた値にして，シミュレーションを行った．

```
*
* Predictive Technology Model Beta Version
* 0.10um NMOS SPICE Parametersv (normal one)
*

.model NMOS NMOS
+Level = 49

+Lint = 2.e-08 Tox = 2.5e-09
+Vth0 = 0.2607 Rdsw = 180

+lmin=1.0e-7 lmax=1.0e-7 wmin=1.0e-7 wmax=1.0e-4
+Tref=27.0   version =3.1
+Xj= 4.0000000E-08       Nch= 9.7000000E+17
+lln= 1.0000000          lwn= 1.0000000              wln= 0.00
+wwn= 0.00               ll= 0.00
+lw= 0.00                lwl= 0.00                   wint= 0.00
+wl= 0.00                ww= 0.00                    wwl= 0.00
+Mobmod=  1              binunit= 2                  xl= 0.00
+xw= 0.00                binflag=  0
+Dwg= 0.00               Dwb= 0.00

+ACM= 0                  ldif=0.00                   hdif=0.00
+rsh= 7                  rd= 0                       rs= 0
+rsc= 0                  rdc= 0
```

```
+K1= 0.3950000           K2= 1.0000000E-02          K3= 0.00
+Dvt0= 1.0000000         Dvt1= 0.4000000            Dvt2= 0.1500000
+Dvt0w= 0.00             Dvt1w= 0.00                Dvt2w= 0.00
+Nlx= 4.8000000E-08      W0= 0.00                   K3b= 0.00
+Ngate= 5.0000000E+20

+Vsat= 1.1000000E+05     Ua= -6.0000000E-10         Ub= 8.0000000E-19
+Uc= -2.9999999E-11
+Prwb= 0.00              Prwg= 0.00                 Wr= 1.0000000
+U0= 1.7999999E-02       A0= 1.1000000              Keta= 4.0000000E-02
+A1= 0.00                A2= 1.0000000              Ags= -1.0000000E-02
+B0= 0.00                B1= 0.00

+Voff= -2.9999999E-02    NFactor= 1.5000000         Cit= 0.00
+Cdsc= 0.00              Cdscb= 0.00                Cdscd= 0.00
+Eta0= 0.1500000         Etab= 0.00                 Dsub= 0.6000000

+Pclm= 0.1000000         Pdiblc1= 1.2000000E-02     Pdiblc2= 7.5000000E-03
+Pdiblcb= -1.3500000E-02 Drout= 2.0000000           Pscbe1= 8.6600000E+08
+Pscbe2= 1.0000000E-20   Pvag= -0.2800000           Delta= 1.0000000E-02
+Alpha0= 0.00            Beta0= 30.0000000

+kt1= -0.3700000         kt2= -4.0000000E-02        At= 5.5000000E+04
+Ute= -1.4800000         Ua1= 9.5829000E-10         Ub1= -3.3473000E-19
+Uc1= 0.00               Kt1l= 4.0000000E-09        Prt= 0.00

+Cj= 0.0015             Mj= 0.72                   Pb= 1.25
+Cjsw= 2E-10            Mjsw= 0.37                 Php= 0.773
+Cjgate= 2E-14           Cta= 0                     Ctp= 0
+Pta= 0                  Ptp= 0                     JS=1.50E-08
+JSW=2.50E-13            N=1.0                      Xti=3.0
+Cgdo=3.493E-10          Cgso=3.493E-10             Cgbo=0.0E+00
+Capmod= 2               NQSMOD= 0                  Elm= 5
+Xpart= 1                cgsl= 0.582E-10            cgdl= 0.582E-10
+ckappa= 0.28            cf= 1.177e-10              clc= 1.0000000E-07
+cle= 0.6000000          Dlc= 2E-08                 Dwc= 0

*
* Predictive Technology Model Beta Version
* 0.10um PMOS SPICE Parametersv (normal one)
*
```

```
.model PMOS PMOS
+Level = 49

+Lint = 2.e-08 Tox = 2.5e-09
+Vth0 = -0.303 Rdsw = 300

+lmin=1.0e-7 lmax=1.0e-7 wmin=1.0e-7 wmax=1.0e-4
+Tref=27.0   version =3.1
+Xj= 4.0000000E-08            Nch= 1.0400000E+18
+lln= 1.0000000               lwn= 0.00                    wln= 0.00
+wwn= 1.0000000               ll= 0.00                     lw= 0.00
+lwl= 0.00                    wint= 0.00                   wl= 0.00
+ww= 0.00                     wwl= 0.00                    Mobmod=  1
+binunit= 2                   xl= 0.00                     xw= 0.00
+binflag=  0                  Dwg= 0.00                    Dwb= 0.00

+ACM= 0                       ldif=0.00                    hdif=0.00
+rsh= 7                       rd= 0                        rs= 0
+rsc= 0                       rdc= 0

+K1= 0.3910000                K2= 1.0000000E-02            K3= 0.00
+Dvt0= 2.6700001              Dvt1= 0.5300000              Dvt2= 5.0000000E-02
+Dvt0w= 0.00                  Dvt1w= 0.00                  Dvt2w= 0.00
+Nlx= 7.5000000E-08           W0= 0.00                     K3b= 0.00
+Ngate= 5.0000000E+20

+Vsat= 1.0500000E+05          Ua= -5.0000000E-10           Ub= 1.5000000E-18
+Uc= -2.9999999E-11
+Prwb= 0.00                   Prwg= 0.00                   Wr= 1.0000000
+U0= 5.5000000E-03            A0= 2.0000000                Keta= 4.0000000E-02
+A1= 0.00                     A2= 0.9900000                Ags= -0.1000000
+B0= 0.00                     B1= 0.00

+Voff= -7.0000000E-02         NFactor= 1.5000000           Cit= 0.00
+Cdsc= 0.00                   Cdscb= 0.00                  Cdscd= 0.00
+Eta0= 0.2500000              Etab= 0.00                   Dsub= 0.8000000

+Pclm= 0.1000000              Pdiblc1= 1.2000000E-02       Pdiblc2= 7.5000000E-03
+Pdiblcb= -1.3500000E-02      Drout= 0.9000000             Pscbe1= 8.6600000E+08
+Pscbe2= 1.0000000E-20        Pvag= -0.2800000             Delta= 1.0100000E-02
+Alpha0= 0.00                 Beta0= 30.0000000

+kt1= -0.3400000              kt2= -5.2700000E-02          At= 0.00
```

+Ute= -1.2300000	Ua1= -8.6300000E-10	Ub1= 2.0000001E-18
+Uc1= 0.00	Kt1l= 4.0000000E-09	Prt= 0.00
+Cj= 0.0015	Mj= 0.7175511	Pb= 1.24859
+Cjsw= 2E-10	Mjsw= 0.3706993	Php= 0.7731149
+Cjgate= 2E-14	Cta= 9.290391E-04	Ctp= 7.456211E-04
+Pta= 1.527748E-03	Ptp= 1.56325E-03	JS=2.50E-08
+JSW=4.00E-13	N=1.0	Xti=3.0
+Cgdo=3.49E-10	Cgso=3.49E-10	Cgbo=0.0E+00
+Capmod= 2	NQSMOD= 0	Elm= 5
+Xpart= 1	cgsl= 0.582E-10	cgdl= 0.582E-10
+ckappa= 0.28	cf= 1.177e-10	clc= 5.4750000E-08
+cle= 6.4600000	Dlc= 2E-08	Dwc= 0

謝辞

本研究を進めるにあたり，終始のご指導と的確な助言を賜わりました本学の日比野 靖教授に心より御礼を申し上げます.

本学の金子 峰雄教授には回路の複雑さの理論についての御助言を頂き、宮地 充子助教授には暗号理論や数学的表記についての御助言を頂きましたことに、深く感謝いたします。

はこだて未来大学 情報アーキテクチャ学科の高木 剛助教授には標数 3 の体について等の御助言を頂きましたことに心より感謝致します.

また，日頃より多くのご指導を頂いた本学の田中 清史助教授，菅原 英子助手に謝意を表します.

最後に，日頃より多くの支援を頂いた計算機アーキテクチャ研究室の皆様に御礼申し上げます.

なお，本論文の成果は文部科学省 21 世紀 COE プログラムによるものです。

参考文献

[1] 宮地充子, 菊池浩明, 「情報セキュリティ」, オーム社, 2003.

[2] A. K. Lenstra and E. R. Verheul, "The XTR Public Key System," *Advances in Cryptology-CRYPTO 2000*, LNCS 1800, Springer, pp.1-20, 2000.

[3] N. Koblitz, "An efficient Curve Implementation of the Finite Field Digital Signature Algorithm," *Advances in Cryptology-CRYPTO '98*, LNCS 1462, Springer, pp.327-337, 1998.

[4] I. Duursma and H. S. Lee, "Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x - d$," *Advances in Cryptology-ASIACRYPT 2003*, LNCS 2894, Springer, pp.111-123, 2003.

[5] T. Kerins, W. P. Marnane, E. M. Popovic and P.S.L.M. Barreto, "Efficient Hardware for the Tate Pairing Calculation in Characteristic Three," *Cryptographic Hardware and Embedded Systems–CHES 2005*, LNCS 3659, Springer, pp.412-426, 2005.

[6] Edgar D. Olson, 公表特許広報 特表 2002-517937 (P2002-517937A)

[7] HSPICE, Synopsys 社

[8] I. F. Blake, G. Seroussi and N. P. Smart, *Elliptic curves in cryptography*, Cambridge University Press, 1999.

[9] D. V. Bailey, C. Paar, "Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms", *Advances in Cryptology–CRYPTO'98*, LNCS 1462, pp472–485, Springer, 1998.

[10] N. Koblitz, (林彬 訳), 「暗号の代数理論」, シュプリンガー・フェアラーク東京, 1999.

[11] R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, Cambridge University Press, 1994.

[12] 永尾 汎,「代数学」, 浅倉書店, 1983.

[13] D. E. Knuth, (渋谷政昭 訳), 準数値算法/乱数, サイエンス社, 1981.

[14] T. Kobayashi, H. Morita, K. Kobayashi and F. Hoshino, "Fast Elliptic Curve Algorithm Combining Frobenius Map and Table Reference to Adapt to Higher Characteristic", *Advances in Cryptology–EUROCRYPT'99*, LNCS 1592, pp176–189, Springer, 1999.

[15] 篠永茂, 藤井吉弘, 野上保之, 森川良孝, "TYPE-II All-One Polynomial Field," SCIS2004, 2004 年暗号と情報セキュリティシンポジウム予稿集 vol.I, pp.377–382

[16] M. Stam and A. K. Lenstra, "Speeding Up XTR," *Advances in Cryptology-ASIACRYPT 2001*, LNCS 2248, Springer, pp.125-143, 2001.

[17] A. K. Lenstra and E. R. Verheul, "Selecting Cryptographic Key Sizes", *Public Key Cryptography*, LNCS 1751, pp446–465, Springer, 2000.

[18] S. Lim, S. Kim, I. Yie, J. Kim and H. Lee, "XTR Extended to $GF(p^{6m})$," *Selected Areas in Cryptography*, LNCS 2259, Springer, pp.301-320, 2001.

[19] PARI/GP, http://pari.math.u-bordeaux.fr/

[20] 特許出願, 日比野 靖, 白勢 政明, 特願 2005-001866, "三値論理関数回路および多値論理関数回路"

[21] 白勢政明, 日比野靖, "CMOS トランスファーゲートによる三値論理回路とその構成法", 多値技報 Vol. MVL-05, No. 1, 2005.

[22] 樋口 龍雄, 亀山 充隆, 「多値情報処理」, 浅昭晃堂 , 1989.

[23] U. Cheng and C. Hu, (三洋電機 (株), (株) 日立製作所 共訳), 「MOSFET のモデリングと BSIM3 ユーザーズガイド」, 丸善, 2002.

[24] 「Star-HSPICE Manual ユーザーズガイド」, NTT アドバンステクノロジ, 2000.

[25] BSIM3 Home Page, http://www-device.eecs.berkeley.edu/〜bsim3/

[26] A. R. Omondi, *Computer Arithmetic Systems*, Prentice Hall, 1994.

[27] J. Silverman, *The Arithmetic of Elliptic Curves*, Springer–Verlag, 1986.

[28] P. L. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Mathematics of computation*, Vol.48, No.177, pp243–264, 1987.

[29] M. Shirase and Y. Hibino, "An architecture for Elliptic Curve Cryptograph Computation", ACM SIGARCH Computer Architecture News, Vol. 33 , Issue 1 (March 2005), pp124-133, October 9, 2004.

発表論文等

学会論文誌

1. Masaaki Hirase, Yasushi Hibino, “XTR over characteristic 3”, IEICE Trans. on Fundamentals, (条件付き採用 2005.12.1., 再投稿日 2006.1.12.).
   Masaaki Hirase, Yasushi Hibino, “A digital signature scheme for XTR”, IEICE Trans. on Fundamentals, (査読中, 投稿日 2005.10.4.).

国際会議発表論文

1. “An architecture for Elliptic Curve Cryptograph Computation”, ACM SIGARCH Computer Architecture News, Vol. 33 , Issue 1 (March 2005) Special issue: Workshop on architectural support for security and anti-virus (WASSA), pp124-133, October 9, 2004.

学会発表

1. 白勢 政明, 日比野 靖, “楕円曲線のためのハードウェアの設計”, 信学技報, Vol. 103, No. 315, ISEC2003-63, 2003.9.
2. 白勢 政明, 日比野 靖, “4 次と 6 次の XTR”, 2004 年暗号と情報セキュリティシンポジウム予稿集, pp.371-376, 2004.1.
3. 白勢 政明, 日比野 靖, “CMOS トランスファーゲートによる三値論理回路とその構成法”, 多値技報, Vol. MLV-05, No. 1, pp.80-89, 2005.1.
4. 日比野 靖, 白勢 政明, “標数 3 の体での XTR”, 信学技報, Vol. 105, No. 51, ISEC2005-3, 2005.5.
5. 白勢 政明, 日比野 靖, “XTR に適したディジタル署名方式”, 信学技報, Vol. 105, No. 395, ISEC2005-96, 2005.11.

特許出願

1. 日比野 靖、白勢 政明, 「三値論理関数回路および多値論理関数回路」(特願 2005-001866)
2. 日比野 靖、白勢 政明, 「CMOS トランスファーゲート論理を用いた三値論理回路とその構成法」(特願 2006-023474)