社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－CSEC－34（5）
2006／7／20

素因数分解ハードウェアの現状（関係式探索ステップ編）

～2006年夏～

伊豆 哲也† 國廣 昇†† 下山 武司†

† 富士通株式会社, 〒211-8588 川崎市中原区上小田中4-1-1
†† 電気通信大学 情報通信工学科, 〒182-8585 調布市調布ヶ丘1-5-1
E-mail: †{izu,shimo-shimo}@jp.fujitsu.com, ††kunihiro@ice.uec.ac.jp

あらまし 素因数分解法アルゴリズムである数体篩法は, RSA暗号に対する脅威として知られている. 数体篩法のうち, 理論的・実験的に最も処理時間を要するのは関係式探索ステップであり, このステップを専用ハードウェアで処理するアプローチが近年になって注目されている. 本稿は, 関係式探索ステップ専用のハードウェアとして提案されたいくつかのデザインの概要をまとめる. また関係式探索ステップハードウェアの実装例についても報告する.
キーワード RSA, 素因数分解ハードウェア, 数体篩法

A Survey on Dedicated Factoring Devices (Sieving Step)

Tetsuya IZU†, Noboru KUNIHIRO††, and Takeshi SHIMOYAMA†

† FUJITSU Limited,
4-1-1, Kamikodanaka, Nakahara-ku, Kawasaki, 211-8588, Japan
†† Dept. of Information and Communication Eng., The University of Electro-Communications,
1-5-1, Choufugaoka, Chofu, 182-8585, Japan
E-mail: †{izu,shimo-shimo}@jp.fujitsu.com, ††kunihiro@ice.uec.ac.jp

Abstract Dedicated factoring devices have attracted much attention since it might be a new threat for RSA. Among the Number Field Sieve method of integer factorization, the relation finding step is the most dominant step in both theory and practice. This article surveys hardware designs for this step: TWINKLE, TWIRL, SHARK, DSH and YASD. Experimental results for this step are also reported.
Key words RSA, dedicated factoring device, Number Field Sieve method

1. はじめに

公開鍵暗号のデファクトスタンダードであるRSA暗号の安全性は, 巨大な合成数の素因数分解が難しいという事実に依存している. 2006年5月時点での分解記録は663ビット[BBF+05]であることから, 現在のRSA暗号が公開鍵として標準的に使用する1024ビット合成数の分解は当面は困難であると考えられているが, この予測は計算機上での素因数分解実験の結果から導出されており, 他の計算アーキテクチャは想定していない. このため, 特にASICベースの素因数分解ハードウェア装置の実現可能性が盛んに議論されるようになっている(*1). これまでの研究成果はASIC実装に適したデザインの提案やFPGA上への実装報告に留まっており, ASICを用いた実験例はまだ知られていない. しかしこれら装置の潜在的な可能性を考慮するならば, 実装可能性の検討は必須である.

数体篩法(Number Field Sieve method)はLenstra, Lenstra Jr., Manasse, Pollardによって1993年に提案された最良の素因数分解法で[LL93], 多項式選択, 関係式探索, 線形代数, 平方根計算の4つのステップから構成される. このうち関係式探索ステップと線形代数ステップは理論的にも現実的にも計算時間のほとんどを占めるため, これらステップに対する素因数分解ハードウェアの実現方法が盛んに議論されている. 本稿は関係式探索ステップに対するハードウェア装置のデザイン例(TWINKLE, TWIRL, SHARK, DSH, YASD)と実装例について現状報告を行う. なお線形代数ステップに対するハードウェアのデザイン例と実装例については, 著者らによる別の報告[IKS06]を参照されたい.

(*1): ECRYPTが(素因数分解ハードウェアを含む)暗号解読用ハードウェアに関する国際ワークショップSHARCSを2005年2月と2006年4月に開催しているのはその一端である.

2. 関係式探索ステップ

本稿では数体篩法における関係式探索ステップの動作原理を簡単に説明する(*2). 整数のペア (a,b) $(-H_a \leqq a \leqq H_a,\ 1 \leqq b \leqq H_b)$ が以下の 3 条件を満たすとき, ペア (a,b) を関係 (relation) と呼ぶ.

- $\gcd(a,b)=1$
- $F_r(a,b)$ は B_r-smooth: ただし 1 次の整数係数 2 変数多項式 $F_r(x,y)=mx+y$, 自然数 B_r は与えられているとする.
- $F_a(a,b)$ は B_a-smooth: ただし d 次の整数係数 2 変数多項式 $F_a(x,y)=(-x)^d f_a(-y/x)$, d 次の既約な整数係数 1 変数多項式 $f_a(z)=c_d z^d+\cdots+c_0$ $(c_i \in \mathbb{Z})$, 自然数 B_a は与えられているとする.

ここで自然数 N が自然数 B に対して B-smooth であるとは, N が B 以下の素因数の積に分解できること, 同じ事だが N が $N=\prod_{p_i|N,\ p_i \leqq B} p_i^{e_i}$ と表せることをいう. なお 2 つ目の条件を有理的 (rational), 3 つ目の条件を代数的 (algebraic) と形容することがある. 2 つ目と 3 つ目の条件は類似しているため, 以下では簡単のため, 多項式 $F(x,y)$ は $F_r(x,y)$ または $F_a(x,y)$ のいずれかを, B は B_r または B_a のいずれかを表すことにする.

関係式探索ステップの目標は, 与えられた 2 次元領域

$$\{\ (a,b) \in \mathbb{Z} \times \mathbb{N} \mid -H_a \leqq a \leqq H_a,\ 1 \leqq b \leqq H_b\ \}$$

から, 上の 3 条件を満たす関係 (a,b) を大量に見つけることである. このとき見つけるべき関係は規定の個数以上であればどのような組み合わせでも良く, また全てを見つける必要もない. 従って関係式探索ステップでの基本的な処理は, 固定された b に対し, $F(a,b)$ が B-smooth となる a を $\{-H_a,\ldots,H_a\}$ の中から見つけ出すことである. もちろん $F(a,b)$ を B 以下の素数 p_i で順番に割っていくことで, $F(a,b)$ が B-smooth であるかを判定することは可能であるが, 以下のように工夫することで効率的な処理が可能となる.

$F(a,b)=\prod_{p_i|F(a,b),\ p_i \leqq B} p_i^{e_i}$ であるとき,

$$\log F(a,b)=\sum_{p_i|F(a,b),\ p_i \leqq B} e_i \log p_i \approx \sum_{p_i|F(a,b),\ p_i \leqq B} \lfloor \log p_i \rceil \qquad (1)$$

と近似することができる (ここで $\lfloor x \rceil$ は実数 x の四捨五入, すなわち x に最も近い整数を表す). このような近似が可能なのは, ほとんどの大きな素因数 p_i に対しては $p_i|F(a,b)$ ならば $e_i=1$ となることと, メモリ効率の観点から対数の値を整数に丸めた方が好ましく, 誤差は後で調整可能なことが理由である.

そこで実装では以下のような処理を行う. 各 a に対応する変数 $s(a)$ (初期値 0) を用意し, 整数 $F(a,b)$ が素数 p_i で割り切れるときに $\lfloor \log p_i \rceil$ を $s(a)$ に加えることにする. この操作を B 以下の素数全てに対して行い, 最終的に $s(a)$ の値が $\log F(a,b)$ に近い場合, (a,b) を関係の候補として扱う. 候補となった (a,b) に対しては, 実際の素因数分解 (ミニ素因数分解) を通じて, 本当に関係になっているかを判定する. なお $s(a)$ の値が $\log F(a,b)$ に近いかどうかの判定は, $s(a)$ の値が近似値であることを考慮して, b によって定まる値 T_b に対し, $s(a)$ が T_b より大きいかどうか判定する. T_b を閾値 (threshold) と呼ぶ. 一般に閾値を緩くすればたくさんの (a,b) が候補として得られ, 候補を見逃す可能性を低くすることができる反面, 関係になっているかの判定に多くの時間を要するようになる.

上の処理は, さらに以下のような高速化が可能である. 多項式 $F(a,b)$ は, $F(a,b)$ が p_i で割り切れるならば, $F(a+p_i,b)$ も p_i で割り切れる, という性質が成立する. そこで素数 p_i について $\lfloor \log p_i \rceil$ を足し込む操作をまとめて行うために, 始めに $(-H_a,b)$, $(-H_a+1,b),\ldots,(H_a,b)$ に対応する $2H_a+1$ 個の変数 $s(-H_a)$, $s(-H_a+1),\ldots,s(H_a)$ (初期値は全て 0) を用意する. 次に素数 p_i に対し $F(-H_a,b)$ から順に p_i で割り切れるかを調べていく. $F(a_0,b)$ で初めて p_i で割り切れるとすると, $F(x,y)$ の性質から $F(a_0+p_i,b)$ も p_i で割り切れることになり, るので, 結局 $F(a_0,b)$, $F(a_0+p_i,b)$, $F(a_0+2p_i,b),\ldots$ は p_i で割り切れる. そこで素数 p_i に対しては, 変数 $s(a_0)$, $s(a_0+p_i)$, $s(a_0+2p_i),\ldots$ に $\lfloor \log p_i \rceil$ を加えることにする. この操作を B 以下の全ての素数 p_i について繰り返していくことで, $F(a,b)$ が B-smooth となる a の候補を $-H_a \leqq a \leqq H_a$ の中からまとめて見つけることができる. このように a をまとめて見つける操作を篩 (sieve) と言う. 篩操作は関係式探索ステップが篩ステップと呼ばれる理由であり, さらには数体篩法の名前の由来にもなっている.

ここまでで説明した篩法は線篩 (line sieve) と呼ばれる手法であるが, 他に格子篩 (lattice sieve) という篩法も知られている. 格子篩は線篩よりも高速処理が可能なことから, 数体篩法のソフトウェア実装では中心的に使用されている. これに対しハードウェアデザインではまだ線篩が中心であるが, 最近になって格子篩の実装が検討され始めている [FKP+05].

3. デザイン例

本節では, 数体篩法の関係式探索ステップを処理するハードウェアのデザイン例として, TWINKLE [Sha99], TWIRL [ST03], DSH [GS03], YASD [GS04] および SHARK [FKP+05] の動作原理を説明する. 以下, 素因数分解したい合成数 N, 2 つの多項式 $F_r(x,y)$, $F_a(x,y)$, 篩の対象範囲を定めるパラメータ H_a, H_b, 因子基底の上限を定めるパラメータ B_r, B_a は与えられているとする.

3.1 TWINKLE

TWINKLE (The Weizemann INstitute Key Locating Engine) は, 関係式探索ステップを線篩アルゴリズムに基づいて処理するハードウェアデザインであり, 1999 年に Adi Shamir によって提案された [Sha99]. しかしデザインの提案に留まっており, これまでにいかなる実装報告もなされていない.

TWINKLE は発光部と受光部に分かれ, 発光部は独立した多数のセル (LED) を持つウェハーとなっている. ここで各セルは素数 p_i に対応しており, 時刻 a のとき, $F(a,b)$ が素数 p_i で割

(*2): 数体篩法の詳細は既存の文献 ([LL93], [IK03] など) を参照されたい.

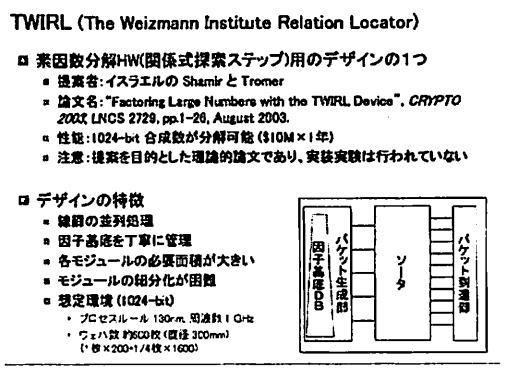


Figure 1 TWIRL の概要

り切れるような p_i に対応するセルが $\lfloor \log p_i \rceil$ に比例する光を放つようになっている. ウェハーの反対側の受光部には (a,b) に対応する変数 $s(a)$ を表す光学素子が設置され, セルから放たれた光を受光する. 最終的な受光量 $\sum \lfloor \log p_i \rceil$ が閾値 T_b を越えた場合, その (a,b) が候補として得られることになる(*3).

TWINKLE の長所は原理が簡単なことである. 各セル (LED) は固有の周期で発光すれば良いので, ハードウェアによる実現も容易である. しかし欠点として, 受光部に光学素子 (GaAs, ヒ化ガリウム) を前提としているため費用が高価であり, さらなる効率化を目的として並列化した場合, 発光部と受光部をそれぞれ複数用意しなければならないことから, 並列化による効果が低いことが挙げられる. このため TWINKLE の性能限界は 512-bit であり, 768-bit の合成数の素因数分解に対応する篩処理を行うことは, 現実的には不可能であると評価されている [LS00]. 半導体素子を用いて TWINKLE を設計することも可能であるが, それでも 512-bit より大きな合成数の処理は困難であると考えられている. これは並列化による効果が低いという TWINKLE の構造的な欠点が原因である. この欠点を克服したのが次節で紹介する TWIRL である.

3.2 TWIRL

TWIRL (The Weizemann Institute Relation Locator) は, 関係式探索ステップを線篩アルゴリズムに基づいて処理するハードウェアデザインであり, 2003 年に Adi Shamir と Eran Tromer によって提案された [ST03]. しかしデザインの提案に留まっており, これまでにいかなる実装報告もなされていない. TWIRL の顕著な特徴は, 1024-bit 合成数の素因数分解に対応する関係式探索ステップの処理が可能と主張されている点である. 具体的には約 10 億円の製造費 (直径 300 mm のウェハ 600 枚) を投資した場合, 約 1 年の計算時間が必要と見積もられている. また 768-bit の場合, 約 22 万円のクラスタを 1 組用いた場合で約 2.3 年, 6 組 (ウェハ 1 枚分) を用いた場合で約 3 ヶ月との見積もりが与えられており, さらに 512-bit の場合では, 約 2 万円のクラスタを 1 組用いた場合で 4.5 時間, 79 組 (ウェハ 1 枚分) を用いた場合で約 3.4 分とされるなど, 他のデバイスに比べて圧倒的な性能の優位性が主張されている. しかし実現可能性・運用の可能性についてはさまざまな指摘・議論があり, 例えば CRYPTREC による評価では, (1024-bit の場合における) 否定的な結果が報告されている [F03], [H03].

(*3) : 各 LED がまたたいている (twinkle) ことが, TWINKLE の真の名称の由来であろう.

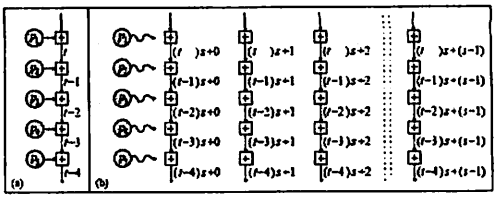


Figure 2 篩位置の配置 (a)=単一の加算器列 (TWINKLE), (b)=複数の加算器列 (TWIRL) [ST03]

1024-bit 合成数の素因数分解に対応する関係式探索ステップを処理する TWIRL のうち, 有理数的篩を処理する TWIRL (Rational TWIRL) についての動作概要を示す. これは TWIRL における Rational TWIRL の占める比重が大きいからである. 以下, 提案者らの推奨値には 2 重括弧 $\langle\langle \cdot \rangle\rangle$ をつけて表し, そのパラメータが Rational TWIRL に関するならば $\langle\langle x \rangle\rangle_r$, Algebraic TWIRL に関するならば $\langle\langle x \rangle\rangle_a$, 両者に関すならば単に $\langle\langle x \rangle\rangle$ のように記す.

TWIRL は, 関係式探索ステップの篩計算を高度に並列処理する点を特徴とする. まず TWINKLE のように 1 clock cycle あたり 1 つの篩位置 a を処理するデバイスを考える. 閾値を T_b とするとき, このデバイスは $\log T_b$ $\langle\langle = 10 \rangle\rangle$ bit 幅の一方向性なデータパスを持ち, 何百万もの条件判断機能付き加算器を連続して持つ. それぞれの条件判断機能付き加算機は 1 つの Progression $P_i = \{\, a = a_0^{(i)} + kp_i \mid k = 0, 1, \dots \,\}$ を担当し, 動作時刻になるとパスに $\lfloor \log p_i \rceil$ を加える. データ $\lfloor \log p_i \rceil$ を p_i に対応するパケットと呼ぶ. 時刻 t の時, z 番目の加算器は $t-z$ 番目の篩位置を処理する. パイプラインには 1 clock cycle あたり 1 つの速さで篩位置のデータが流れ, 0 を先頭に $1, \dots, 2H_a$ が流れていく (Figure 2 (a) 参照).

次にこのデバイスの並列化を行う. 先ほどと同じ $\log T_b$ bit 幅のパスを s $\langle\langle = 4,096 \rangle\rangle_r$, $\langle\langle = 32,768 \rangle\rangle_a$ 本用意し, 時刻 t の時には z 番目の加算器の i 本目のパスは $(t-z)s+i$ 番目の篩位置を処理するように変更する. ここで各パイプラインの先頭になる篩位置を $0, 1, \dots, s-1$ とする (Figure 2 (b) 参照). このように変更することで, 1 clock cycle あたりに処理するデータを s 個にすることが可能となる. ただし 2 つの問題が生じてしまう. 1 つは時刻の進みが $1/s$ になるため, それに応じた処理が必要になること, もう 1 つは s 本のパスを同時に並列処理するために, 同じ Progression P_i に対応する加算 $\lfloor \log p_i \rceil$ を同時に行う必要が生じることである(*4). そこで同じ Progression P_i を担当する送信器を複数用意し, 素数の頻度に合わせて 3

(*4) : もちろん Progression 送信器を s 組用意すれば良いが, それでは s 個の TWINKLE を同時に処理しているのと同等の効果しか得られず, TWIRL のメリットが消えてしまう.

種類の Station と呼ばれるユニットを用いる．具体的には，大きな素数に対しては Largish Station，小さな素数に対しては Smallish Station，とても小さな素数に対しては Tiny Station を用いる．各 Station は決められた範囲の素数を担当し，パイプライン状に連結されている．最後の Station の出力後に閾値判定を行うユニットを配置し，その出力をデバイスの出力とする．Largish/Smallish/Tiny Station の詳細な設計については提案論文 [ST03] を参照されたい(*5)．

関係式探索装置としての TWIRL は，複数個の Rational TWIRL が 1 個の Algebraic TWIRL に連結される構造となっている．1024-bit 合成数に対しては，《8》個の Rational TWIRL が 1 個の Algebraic TWIRL に連結されている．

次に，1024-bit 合成数に対応する TWIRL のコスト見積もりを述べる(*6)．提案者等によると，1 個の Rational TWIRL が必要とするシリコンの面積は 15,960 mm^2 で，このうち Largish Station は 76 % (DRAM は全体の 37 %)，Smallish Station は 21 %，Tiny Station は残りである 3 % を占める．同様に，1 つの Algebraic TWIRL が必要とするシリコンの面積は 65,900 mm^2 で，このうち Largish Station は 94 % (DRAM は全体の 66 %)，Smallish Station は 6 % を占める．1 組の TWIRL クラスタは，8 個の Rational TWIRL と 1 個の Algebraic TWIRL から構成され，全体で必要なシリコンの面積はウェハー 3 枚となる．各 Rational TWIRL は Algebraic TWIRL に対して単方向的接続をしていて，13 bit/clock cycle でデータを転送する．クラスタは 1 本のラインをふるうのに，スループットで $2H_a/s_A$ clock cycle (周波数 1GHz で約 33.4 秒) を必要とするので，全領域 (H_b 本のライン) をふるうには，約 286 年 ($33.4 \times H_b = 9.02 \times 10^9$ 秒) が必要である．さらに約 33 % の高速化が可能であることから，約 194 年 ($= 286 \times 0.67$) に改善できる．従って 194 組のクラスタを用いれば，約 1 年で篩処理が終了する．ウェハー 1 枚の製造コストを 5,000 ドルと仮定した場合，このときの製造コストは合計で約 290 万ドルとなる．本節の冒頭で述べた通り，このような見積もりに関しては，さまざまな指摘・議論がなされている．

3.3 SHARK

SHARK は，関係式探索ステップを格子篩アルゴリズムに基づいて処理するハードウェアデザインであり，2005 年に Jens Franke, Thorsten Kleinjung, Christof Paar, Jan Pelzl, Christine Priplata, Colin Stahlke によって提案された [FKP+05]．しかしデザインの提案に留まっており，これまでにいかなる実装報告もなされていない．SHARK の顕著な特徴は 1024-bit 合成数の素因数分解に対応する関係式探索ステップの処理が可能とされている点であり，具体的には約 92 億円の製造費を投資した場合(*7)，約 1 年の計算時間が必要であると見積もられ

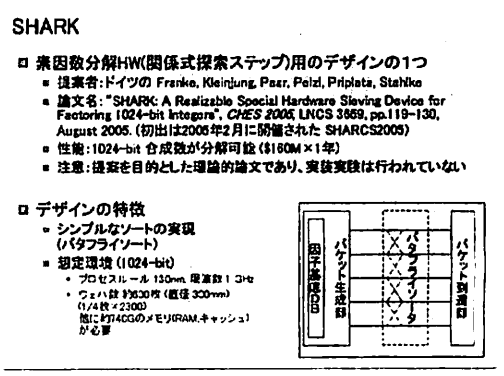


Figure 3 SHARK の概要

ている．TWIRL (の 1024-bit 版) に比べて約 10 倍の製造費を必要としているが，SHARK は TWIRL に比べ高い実現可能性を持っていると主張されている．

まず格子篩について簡単に説明する．関係式探索ステップの目標は $F(a,b)$ が素数の積で表せるような関係 (a,b) を見つけることだが，このような条件を満たす確率は非常に小さいため，近い性質を持つ (a,b) から関係を構成することを考える．そこで線篩は $F(a,b)$ のほとんどの素因数が因子基底に含まれ，数個 (2〜3 個) の素因数が含まれない場合を再利用し，例外の素因数が soothness bound B の数倍 (例えば 10 倍) 程度であれば，因子基底にこれらの素数を追加し，その (a,b) が smooth になるように因子基底を変更する．このようなテクニックを large prime variation と呼ぶ．線篩では，$F(a_0,b)$ がある素数 p_i で割り切れるならば $F(a_0+p_i,b)$ も素数 p_i で割り切れるという性質が用いられていたが，実はこの性質は b についても成立し，$F(a,b_0)$ がある素数 p_i で割り切れるならば $F(a,b_0+p_i)$ も素数 p_i で割り切れる．つまり $F(a,b)$ が素数 p_i で割り切れるような (a,b) の集合は 2 次元のベクトル空間，特に格子を生成しており，格子篩はこのような格子構造，つまり因子基底に含まれないような素数 q に対し，$F(a,b)$ が q で割り切れるような (a,b) が持つ格子構造を利用するこの工夫を special-q と呼び，上記の large prime variation が効率的に実現できる．なお，与えられた B に対して，合成数が小さくなるにつれてその合成数が smooth になる確率は増加する．線篩では $|F(a,b)|$-bit の合成数が smooth になる場合を考えていたが，格子篩では $|F(a,b)/q|$-bit の合成数を考えることになるので，smooth になるペア (a,b) を効果的に見つけられるのである．

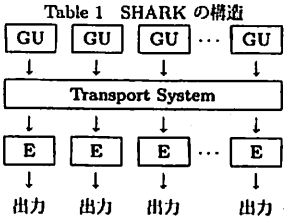
SHARK の構造は以下の通りである (Table 1 参照)．1 組の SHARK は，パケット生成部 (GU; Generation Unit)，ソータ部 (Transport System)，パケット到達部 (Evaluator) の 3 つのパートから構成される．GU は因子基底を保持するための 64MB の DRAM を持ち，巨大な素数 (2^{22} より大きな素数) に

(*5)：Largish Station の因子基底格納用メモリが環状に設計され，回転 (twirl) しながら packet を生成することが，TWIRL の真の名称の由来であろう．

(*6)：なお提案者等はプロセスルールの変更や数体篩法における多項式ステップの改良により，約 10 倍のコストダウンが可能であることが主張されているが [LTS+03]，これら変更・改良による効果は他のデザインに対しても同様であることから，以下の見積もりでは考慮していない．

(*7)：提案論文ではこの他にも電力供給機・冷却機の製造に約 68 億円，1 年間の消費電力に約 60 億円が必要で，総コストは 220 億円と見積もられている [FKP+05]．他の素因数分解ハードウェアデザインではこれらの見積もりは考慮されていないため，簡単のため本稿ではこれらの費用を除外する．

Table 1 SHARK の構造



関するパケットを生成する. Transport System とはソータの一種であるが, Butterfly ソートと呼ばれる簡単なソーティング機能だけを持つ. Evaluator は小さな素数に対応するパケット生成機能, 各 (a,b) に対応する変数 $S(a,b)$ を保持するためのメモリ (32MB), メモリの内容から関係の候補となる (a,b) を判定する機能を持つ. 候補と判定されたペア (a,b) が本当に関係となっているかを判定するために, Evaluator にはさらにミニ素因数分解を処理する LSI が接続されている. 1 つの Transport System に対し 1024 組ずつの GU と Evaluator が接続されている.

SHARK の主たる目標は, GU で生成されたパケットを Evaluator 内の適切なメモリに送ることである. この目標を達成するために, TWIRL はハードウェアによるソータを利用したが, スループットを確保するためには巨大なソータが必要になるという問題が生じていた. そこで SHARK はバケツソーティングと呼ばれるソーティングを用いることで, ソータの回路規模を抑制している. バケツソーティングとは, データを 2 段階に分けてソートするテクニックで, 例えば 1 回目のソーティングではデータの上位 10-bit を用いて処理し (バケツへのソーティング), 2 回目のソーティングでは残りの bit を用いて処理する (各バケツ内のソーティング) 技法である. SHARK では, 1 回目のソーティングが Butterfly ソーティング, 2 回目のソーティングが Evaluator 内のソーティングに対応する.

Butterfly ソーティングとは, 簡単な結線だけでソーティングを実現する方法である (Figure 3). 2^n 入力, 2^n 出力の Butterfly ソーティングでは, $2^n \times (n+1)$ のバッファを用意し, 各バッファは 2 入力, 2 出力となるように結線する. ただし上下のバッファとは必ず結線することとし, もう 1 つの入出力バッファはあるルールに従って結線する. あるバッファに入力されたパケットは, 2 つのうちのいずれかのバッファに送信される. ここで入出力ルールをうまく設定することで, 各バッファへの入力が常に 1 つになるようにコントロール可能である. このようにして GU で生成されたパケットは Butterfly ソーティングによって適切な Evaluator に送信されることになる.

提案者らによる SHARK のコスト見積もりを述べる. 概算によると, 1 組の SHARK デバイスは 136 GB の DRAM と 192 MB のキャッシュ (約 210 万円), プロセッサとしてウェハ 1.8 枚程度のサイズの ASIC (約 90 万円), 制御用の PC (約 100 万円) を必要とする. 1024-bit 合成数に対応する関係式探索ステップを処理する場合, 約 1 年で処理を終えるには 2300 組の SHARK が必要であり, 製造費用として約 92 億円を必要とする. この他に電力供給機・冷却機の製造に約 68 億円, SHARK を実際に 1 年間作動させた場合の消費電力に約 60 億円が必要で, トータルのコストは約 220 億円と見積もられている (さらには初期製造コストも必要であろう). SHARK のコア部分の製造費を TWIRL の製造費と比べると, 約 10 倍の費用を必要としていることがわかる. これはデバイスの製造可能性を高めるため, TWIRL のような (非現実的な) ソータではなく, Butterfly ソータという効率は劣るが実現性の高いソータを利用したからであると思われる.

3.4 DSH

DSH (Dedicated Sieving Hardware) は, 関係式探索ステップを線篩に基づいて処理するハードウェアデザインであり, 2003 年に Willi Geiselmann と Rainer Steinwandt によって提案された [GS03]. しかしデザインの提案に留まっており, これまでにいかなる実装報告もなされていない. 提案者らの評価によると, DSH の性能限界は 512-bit であるとされ, メッシュサイズは $m = 2^{11}$ で, 直径 300 mm のウェア 1 枚に収まるサイズと見積もられている. このとき, サイズ $S = 2^{22}$ の篩区間を 32×2^{11} クロックで処理可能であり, 関係式探索ステップ全体では 4 日程度を必要とすると主張されている. DSH の特徴は, パケットの送信制御にソーティングを用いる点であり, Bernstein が線形代数ステップ用のハードウェアデザインにおいて提案したアイディア [Ber01] を, 関係式ステップに拡張したと見なすことができる. ただし次節で紹介する YASD の方が効率的に優れていると考えられている.

512-bit 合成数の素因数分解に対応する関係式探索ステップを処理する DSH の処理概要を以下に示す. ここで因子基底

$$P_r = \{\, (p,r) \mid p | F_r(-H_a + r, 1),\ 2^{22} < p < 2^{24},\ 0 \leqq r < p \,\},$$
$$P_a = \{\, (p,r) \mid p | F_a(-H_a + r, 1),\ 2^{22} < p < 2^{24},\ 0 \leqq r < p \,\}$$

はあらかじめ与えられているとする. また簡単のため, メッシュのサイズ m は $m^2 = \#P_r + \#P_a$ を満たすとする. ここで $\#P$ は集合 P の要素数を表す.

このとき DSH の処理概要は以下の通りである:

(0) (初期化) $\#P_r + \#P_a$ 個のデータ (p,r,i) $(i \in \{\mathrm{r},\mathrm{a}\})$ を各ノードに格納する. また $b \leftarrow 1$ とする.

(1) 区間 $[-H_a, H_a]$ 内の幅 $S = 2^{22}$ の区間に対する以下の篩操作を行う.

(a) データ (p,r,i) を蛇上に小さい順にソートする. ここで大小関係は $(p_0,r_0,i_0) < (p_1,r_1,i_1) \Leftrightarrow r_0 || i_0 < r_1 || i_1$ と定める. ただし記号 $||$ は接合 (concatenation), $\mathrm{r} < \mathrm{a}$ と定める. ソーティングの結果, データは以下のように並ぶとする.

$$\ldots, (p', r', \mathrm{a}),\ (p_1^{\mathrm{r}}, r, \mathrm{r}), \ldots,\ (p_{\ell_r}^{\mathrm{r}}, r, \mathrm{r}),\ (p_1^{\mathrm{a}}, r, \mathrm{a}), \ldots,$$
$$(p_{\ell_a}^{\mathrm{a}}, r, \mathrm{a}),\ (p'', r'', \mathrm{r}), \ldots$$

データ $(p_{\ell_r}^{\mathrm{r}}, r, \mathrm{r})$, $(p_1^{\mathrm{a}}, r, \mathrm{a})$ を保持している隣接したノードを Q_r^*, Q_a^* と呼ぶ.

(b) 各ノードは, カウンタ c を $c \leftarrow \lfloor \log_2 p \rfloor$ と設定する.

(c) 各ノードは両隣のノードに $r||i$ を送信/受信し, その

ノードが「同一の r, i を持つノードの中で端かどうか」を判定する (ただし $i = \mathrm{r}$ であるものは後端かどうかを, $i = \mathrm{a}$ であるものは前端かどうかを判定する).

(d) 端でないノードは, $i = \mathrm{r}$ ならばカウンタ値を後方へ, $i = \mathrm{a}$ ならばカウンタ値を前方へ送り, 値を受け取ったノードは自分のカウンタをその値で更新する. 端のノード Q_r^*, Q_a^* は受け取った値をカウンタに足し込む. この操作を全ての $\lfloor \log p \rfloor$ が足されるまで繰り返す. ここまでの処理により, ノード Q_r^* のカウンタには $s_r = \sum_{j=1}^{\ell_r} \lfloor \log p_j^r \rfloor$ が, ノード Q_a^* のカウンタには $s_a = \sum_{j=1}^{\ell_a} \lfloor \log p_j^a \rfloor$ が格納される.

(e) ノード Q_r^* は $s_r > T_r$ であるかを, ノード Q_a^* は $s_a > T_a$ であるかを判定し, 成立する場合には OK フラグを立てる. 次にこれらのノード間に OK フラグの値を交換し, それらの AND をとった結果を OK フラグに格納する.

(f) ノード Q_r^* は OK フラグの値を前方に, ノード Q_a^* は OK フラグの値を後方に送る.

(g) データ (p, r, i, OK) を大きい順にソートする. ここで大小関係は $(p_0, r_0, i_0, \mathrm{OK}_0) < (p_1, r_1, i_1, \mathrm{OK}_1) \Leftrightarrow \mathrm{OK}_0 \| r_0 < \mathrm{OK}_1 \| r_1$ と定める. ソートの結果, OK フラグが 1 であるものが前方に集まる. 前方からデータ (p, r, i) を取り出す. $a = -H_a + (u-1)S + r$ とすると, (a, b) は関係の候補であり, $F_i(a, b)$ を精密に検査する (ここで篩の際に発見した素因数の情報は全て保持されている).

(h) (次のステップへの移行) 各ノードの r の値を $r - S$ に置き換え, この値が負の場合には, さらに p を足し込む. また OK フラグを 0 クリヤする. そして Step 1.(a) に戻る. この処理を $2H_a/S$ 回繰り返す.

(2) $b \leftarrow b+1$ とし, 各ノードにデータ $(p, (r - H_a)b + H_a \bmod p, i)$ を格納した後に Step 1 へ戻る.

ここまでの説明では $2^{22} < p < 2^{24}$ を満たす素数しか扱わなかった. そこで上の処理を $2^{17} < p < 2^{24}$ の場合に拡張する. そのため, ノードに格納する値を $(k \cdot p, r + (\ell - 1) \cdot p, i)$ に変更する. ただし $1 \leq \ell \leq k = \lceil S/p \rceil$ とする ($2^{22} < p$ の場合には何も変更されていない). このとき, 上のアルゴリズムによって $2^{17} < p < 2^{22}$ の素数も正しく篩処理が行われることが簡単な計算からわかる. $2^{17} < p < 2^{24}$ で考えたとき $\#P_a = 2{,}025{,}624 \approx 2^{21}$ であり, $\#P_r$ も同程度になることから, このとき $\#P_r + \#P_a \approx 2^{22}$ となる.

残りの $p < 2^{17}$ を満たす素数については, 上のような篩処理は行わず, 試行除算を用いるものとする.

512-bit 合成数に対応する関係式探索ステップを処理する DSH の提案者らによるコスト見積もりを述べる [GS03]. メッシュのサイズを $m = 2^{11}$ とするとき, 1 つの篩区間に対する篩処理では, 全てのノードにデータをロードするのに $4m$ クロック, 1 回の Schimmler ソーティングに $16m$ クロックが必要であるとし, 他の操作に要する時間は無視する. このとき全体で必要なクロック数は $20m$ クロックとなる. 篩区間の長さを $S = 2^{22}$ とすると, $2H_a/S \approx 4292$ 回の篩が必要で, 全体では $20m \times 2H_a/S \approx 2^{27.4}$ クロックが必要となる. $H_b = 9 \times 10^5$ とし, このデバイスの周波数を 500 MHz とすれば, 全体の篩に

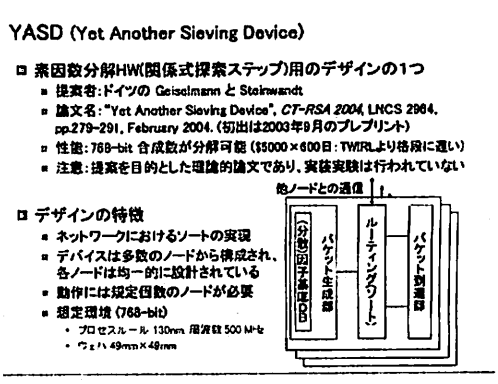


Figure 4 YASD の概要

必要な時間は約 4 日となる. また 1 個のノードは 2,500 個程度のトランジスタで実現可能であり, $m \times m$ のメッシュは直径 300 mm のウェハ 1 枚に収まる大きさとなる.

3.5 YASD

YASD (Yet Another Sieving Device) は, 関係式探索ステップを線篩に基づいて処理するハードウェアデザインであり, 2004 年に Willi Geiselmann と Rainer Steinwandt によって提案された [GS04]. しかしデザインの提案に留まっており, これまでにいかなる実装報告もなされていない. 提案者らの評価によると, YASD の限界性能は 768-bit であるとされている. このときのメッシュサイズは $m = 2^8$ で, 1 組のメッシュは約 49 mm × 49mm で実装可能であると見積もられている. 直径 300 mm のウェハ 1 枚分のチップ (21 組) を用いた場合, 周波数 500 MHz では, 768-bit 合成数の素因数分解に対応する関係式探索ステップの処理に約 600 日を必要すると主張されているこれは TWIRL (の 768-bit 版) に比べ, 同じコストをかけた場合に, 約 6.3 倍遅い結果となっている.

768-bit 合成数の素因数分解に対応する関係式探索ステップを処理する YASD の処理概要を説明する. メッシュのサイズは $m = 2^8$ とし, 篩の探索範囲を $-H_a \leq a \leq H_a = 1.7 \times 10^{13}$, $1 \leq b \leq H_b = 8.9 \times 10^6$ とする. b の値は固定され, a は幅 $S = 2^{24}$ の部分区間に分割されて探索される. なお YASD では, 因子基底に用いる素数を以下のような 4 つのグループに分割して扱う: Tiny prime: $2 \leq p < 2^{17}$, Smallish prime: $2^{17} \leq p < 2^{24} = S$, Largish prime: $2^{24} \leq p < 10^8 = B_r$, Hugish prime: $10^8 \leq p < 10^9 = B_a$.

各部分区間に対する篩処理について説明する. 篩処理の目的は, 篩位置 $\bar{a} \in [s_0, s_0 + S - 1]$ と因子基底情報 (p, r_p) に対し, $\bar{a} \equiv br_p \pmod p$ となるような $\bar{a}$ に対応するレジスタ $s_i(\bar{a})$ $(i \in \{\mathrm{r}, \mathrm{a}\})$ に $\lceil \log p \rceil$ を加えていくことである. 各プロセッサはメインパート, メッシュパート, メモリーパートの 3 つのパートから (概念的に) 構成される. ここでメインパートは因子基底からターゲットノードを決定し, メッシュパートはルーティングを処理, メモリーパートは到着した対数値を蓄積する. 以下パート別に機能を説明する.

（1） メインパート: メインパートは因子基底に関連する情報を保管する DRAM と, その読み書きに関係する回路から構成される. 特にこの回路は有理的な解と代数的な解を区別するフラグを判定できる. またメインパートには各素数の対数値 $\lceil \log(p) \rceil$ (6-bit) も保管される.

メインパートが DRAM から因子基底情報 (p, r) を読み込むと, 始めにその r が処理中の部分区間に対して適切であるかをチェックし, r が適切でない場合には $r \leftarrow (r - S) \bmod p$ によって次の部分区間のためにパラメータを変更しておく(*8). ここで $S \bmod p$ の値が必要となるので, Tiny/Smallish prime に対してはこの値をあらかじめ因子基底に保持しておく. 他方で Largish/Hugish prime に対しては $S < p$ となることから $S \bmod p = S$ なので, 値を保持する必要はない.

r が適切ならば, この r を担当するターゲットノードの座標 (x_t, y_t) を求める. また境界を越えるかどうかを判定するフラグ (c_x, c_y) も求める. パケットは以下のような 59-bit の情報として扱われる: 座標値 x_t, y_t (8-bit ずつ), 境界フラグ c_x, c_y (1-bit ずつ), p の 'footprint' (26-bit), $\lceil \log(p) \rceil$ の値 (6-bit), r の下位 8-bit, 代数側と有理数側のどちらの情報であるかを判定するフラグ (1-bit) である. これら 59-bit の情報がメッシュパートへ送られる. 提案者らの実験では, 出力用のバッファには 59-bit レジスタを 2 つ用意すれば十分とされている.

p の 'footprint' とは, 対応する値の素因数分解情報であって, ターゲットノード情報 (16-bit) に p の下位 10-bit (ただし下位 0-bit 目は 1 であるので, 下位 1-bit 目から下位 10 ビット目とする) を添付したものである (ただし p は閾値 $B_f = 2^{22}$ より大きい場合に限る)(*9). 各ノードが保管する footprint の個数は 850 個程度なので, ほとんどの場合 footprint から p の情報を復元することが可能となる.

ターゲットノードが自分自身である場合には座標情報は必要ないので, パケット情報は 41-bit で済む. この情報はメッシュパートに送る必要はないので, メモリーパートの 41-bit 幅の入力バッファに直接送られることになる.

パケットの生成が終了したらば, 次に r の更新 $r \leftarrow r + p$ を行う. 新しい r の値が妥当であれば, 上記の手続きを繰り返す. そうでなければ r を更新して次の部分区間の処理に備える.

なお各ノードが保持する因子基底のほとんどの素数は Hugish prime であり, Hugish prime を保管するには 38-bit 程度が必要となるため, DRAM は最低でも $38 \times 1{,}300 \approx 50{,}000$-bit が必要である. 提案論文では 55,000-bit が必要とされている. なおメインパートは 2,750 transistor 程度が必要とされている (DRAM を除く).

（2） メッシュパート:

メッシュパートはルーティング (時計回り置換) を行うための回路が置かれている. メッシュパートはメッシュ上を移動するパケットを保持するためのレジスタを 1 つ持つ. このレジスタはメインパートの出力バッファと同じ幅 (59-bit) であり, パケットがターゲットノードに到着した際には, そのパケットから座標情報を除いた 41-bit 情報がメモリーパートの入力バッファに送られる. メッシュパートは 1,100 transistor 程度が必要とされている.

（3） メモリーパート:

メモリーパートは, 各篩位置 a に対して 2 種類の 10-bit DRAM を用意し, 有理側と代数側のそれぞれの対数の積算値 $\lceil \log(p) \rceil$ を保持する. これらレジスタの初期値は 0 であり, 部分区間が更新されるまで値は蓄積される. 各ノードは 256 個の篩位置に関する対数値を蓄積するので, $256 \times 2 \times 10 \approx 6{,}000$-bit 程度の DRAM が必要となる.

メモリーパートは 41-bit 幅の入力バッファから篩位置に関する情報を読み込み, 対応するレジスタに $\lceil \log(p) \rceil$ の値を加える. そして各 10-bit 値とは別に用意された DRAM に footprint 情報を保管する. 実験では 41-bit 幅の入力バッファは 1 つで十分だったと述べられている. footprint を保管するには, footprint 自体が 26-bit の情報であることに加え, 篩位置を特定するために 8-bit 情報が, 代数側か有理数側かを特定するために 1-bit フラグが必要となるので, 合計で 35-bit が必要となる. YASD の提案論文では footprint 用の DRAM は 2 つのノードで共有するのが効率的であると主張されている. このとき, footprint がどちらのノードに関する情報かを示す 1-bit フラグを追加して, footprint は 36-bit 情報として表されることになる. 実験では, ノード 2 つに対する footprint の記憶領域は 325 個分 $(325 \times 36 \approx 11{,}000$-bit) で十分であった. なおメモリーパートは 1,250 transistor 程度が必要とされている (DRAM を除く).

幅 S の部分区間の篩処理を終えた後に, 関係式の候補となる篩位置 a の情報 (24-bit/個) が出力される. また付随する footprint 情報 (26-bit/個) および付随情報 (9-bit/個) も出力される.

768-bit 合成数の素因数分解に対応する関係式探索ステップを処理する YASD の提案者らによるコスト見積もりを述べる [GS04]. $m = 2^8$ としたときに, メッシュ 1 組あたりに必要な回路面積は 49 mm × 49 mm と見積もられ, 直径 300 mm のウェハ 1 枚からは 21 組のメッシュが製造可能とされている. 動作周波数を 500 MHz とするとき, 関係式探索ステップを処理するには約 600 日が必要であると主張されている. これは TWIRL (の 768-bit 版) に比べ, 同じコストをかけた場合に, 約 6.3 倍遅い結果となっている.

YASD は 768-bit 合成数の処理を目的としているため, 提案論文で用いられているパラメータでは 1024-bit 合成数を処理することができず, 拡張の可能性検討は課題とされていた. この課題に対し, 廣田, 伊豆, 國廣, 太田は YASD を任意のビット長の合成数に適用した場合を考察し, パラメータの導出式を示した [HIK+06]. 特に YASD によって 1024-bit 合成数を処理

(*8) : 提案論文では

$$r := \begin{cases} r - (S \bmod p) & \text{if } r - (S \bmod p) \geqq 0 \\ r - (S \bmod p) + p & \text{otherwise} \end{cases}$$

となっている.

(*9) : footprint には $S = 2^{24}$ 個の篩位置を特定する情報 24-bit と, 素数情報 10-bit と, 代数側/有理数側を判定するフラグ 1-bit が必要となるため, footprint の最終的なサイズは 35-bit となる.

Table 2 関係式探索ステップハードウェアの比較

		1024-bit	756-bit
TWIRL	回路規模	$15960mm^2 \times 8 + 66000mm^2$	$1330mm^2 + 4430mm^2$
(1GHz)	時間/費用	194 年 / 15000 USD	2.3 年 / 750 USD
SHARK	回路規模	1/4 wafer + DRAM	(評価なし)
(1GHz)	時間/費用	2300 年 / 40000 USD	
YASD	回路規模	$42200mm^2$	$2400mm^2$
(500MHz)	時間/費用	10652 年 / 3200 USD	34.5 年 / 250 USD

した場合の見積もりがなされており, 約 10 億円の製造費をかけた場合, 周波数 500 MHz では約 16 年を必要とすることが報告されている. これは TWIRL (の 1024-bit 版) に比べ, 同じコストをかけた場合に, 約 16 倍遅い結果となっている.

3.6 比　　較

本稿で紹介した関係式探索ステップ用のハードウェアデザインのうち, TWIRL, SHARK, YASD に関する評価結果を Table 2 にまとめる. なお費用は製造コストだけを考慮しており, 初期開発費用・人件費・歩留まり・電源回路等のコストは除外されている.

4. ま と め

本稿は数体篩法の関係式探索ステップに対する素因数分解ハードウェアの処理アルゴリズムとコストについてまとめた. このようなハードウェアに関する理論的な検討は着実に進んでいるのに対し, 実装実験については, いかなる結果も報告されていない点に注意が必要である. これは, いずれのデザインも回路規模が巨大であり, 開発には多大な費用・コストが必要となるからと考えられる. 従って, 素因数分解ハードウェアの脅威を評価するには, さらなる議論が必要であろう.

なお関係式探索ステップ専用ハードウェアの製造可能性を評価する第一歩として, 下山, 伊豆, 小暮は DAPDNA2 というリコンフィギュアラブルプロセッサ上で, 線篩に基づく関係式探索ステップの実装結果を報告している [SIK06]. 本実装の性能限界は 330 ビット (100 桁) であるものの, 実際の実装・実験を成功させていることなど, PC 以外のプラットフォームにおける実装例という点で非常に興味深い.

謝辞　本研究成果の一部は, 独立行政法人情報通信機構 (NICT) の平成 18 年度委託研究「素因数分解の困難性に基づく技術評価に関する研究開発」の支援のもとで行われました。

References

[Ber01] D. Bernstein, "Circuits for integer factorization: a proposal", preprint, 2001. http://cr.yp.to/papers/nfscircuit.pdf

[BBF+05] F. Bahr, M. Boehm, J. Franke, and T. Kleinjung, "RSA200", May 2005. http://www.crypto-world.com/announcements/rsa200.txt

[F03] 富士通株式会社, "素因数分解装置の調査・検討に関する報告書", 暗号技術関連の調査報告 (2003 年度), 情報処理振興事業協会 (IPA), 通信・放送機構 (TAO), 2004 年 2 月. http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/documents/rep_ID0208.pdf

[FKP+05] J. Franke, T. Kleinjung, C. Paar, J. Pelzl, C. Priplata, and C. Stahlke, "SHARK: A Realizable Special Hardware Sieving Device for Factoring 1024-bit Integers", *CHES 2005*, LNCS pp.27-37, 2005.

[GS03] W. Geiselmann, and R. Steinwandt, "A dedicated sieving hardware", *PKC 2003*, LNCS 2567, pp.254-266, Springer-Verlag, 2003.

[GS04] W. Geiselmann, and R. Steinwandt, "Yet another sieveing device", *CT-RSA 2004*, LNCS 2964, pp.278–291, Springer-Verlag, 2004.

[H03] 株式会社日立製作所, "素因数分解専用集積回路等の実現性についての評価 " 暗号技術関連の調査報告 (2003 年度), 情報処理振興事業協会 (IPA), 通信・放送機構 (TAO), 2004 年 2 月. http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/documents/rep_ID0209.pdf

[HIK+06] N. Hirota, T. Izu, N. Kunihiro, and K. Ohta, "An Evaluation of the Sieving Device YASD for 1024-bit Integers (Extended Abstract)", 2nd Workshop for Special-purpose Hardware for Attacking Cryptographic Systems (SHARCS 2006), ECRYPT, April 2006.

[IK03] 伊豆 哲也, 木田 祐司, "素因数分解の現状について", 日本応用数理学会論文誌 Vol. 13, No. 2, pp.289-304, 2003.

[IKS06] 伊豆 哲也, 國廣 昇, 下山 武司, "素因数分解ハードウェアの現状 (線形代数ステップ編) ", *DICOMO 2006*, 2006 年 7 月.

[LL93] A. Lenstra, and H. Lenstra, editors. The development of the number field sieve, Vol. 1554 of Lecture Notes in Mathematics (LNM), Springer-Verlag, 1993.

[LLP+90] A. Lenstra, H. Lenstra, M. Manasse, and J. Pollard, "The Number Field Sieve", *STOC 1990*, pp.564-572, 1990.

[LS00] A. Lenstra, and A. Shamir, "Analysis and optimization of the TWINKLE factoring device", *EUROCRYPT 2000*, LNCS 1807, pp.35-52, Springer-Verlag, 2000.

[LST+02] A. Lenstra, A. Shamir, J. Tomlinson, and E. Tromer, "Analysis of Bernstein's circuit", *ASIACRYPT 2002*, LNCS 2501, pp.1–26, Springer-Verlag, 2002.

[LTS+03] Arjen K. Lenstra, Eran Tromer, Adi Shamir, Wil Kortsmit, Bruce Dodson, James Hughes, and Paul Leyland, "Factoring estimates for a 1024-bit RSA modulus", *ASIACRYPT 2003*, LNCS 2894, pp.55-74, Springer-Verlag, 2003.

[Sha99] A. Shamir, "Factoring large numbers with the TWINKLE device (extended abstract)", *CHES 1999*, LNCS 1717, pp.2-12, Springer-Verlag, 1999.

[SIK06] 下山 武司, 伊豆 哲也, 小暮 淳, "数体篩法による素因数分解アルゴリズムのハードウェア DAPDNA2 への実装", 2006 年暗号と情報セキュリティシンポジウム (SCIS2006), 2E1-2, January 2006.

[ST03] A. Shamir, and E. Tromer, "Factoring large numbers with the TWIRL device", *CRYPTO 2003*, LNCS 2729, pp.1-26, Springer-Verlag, 2003.