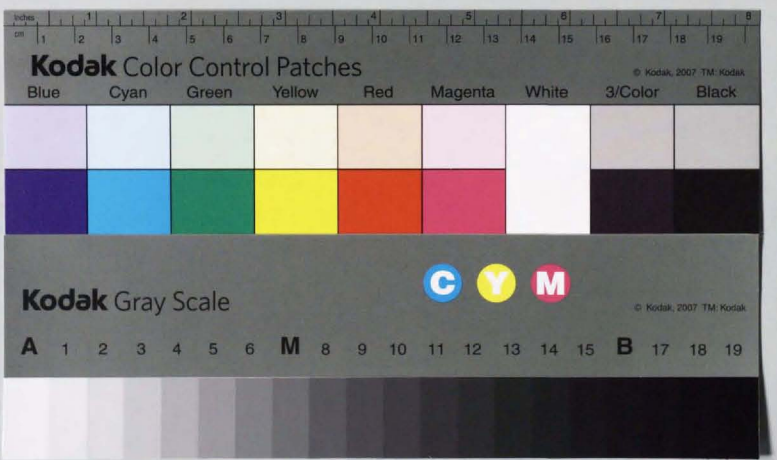

べき乗剰余演算の依頼計算と
高速化に関する研究
川島貴一


Kodak Color Control Patches
© Kodak, 2007 TM: Kodak
Blue
Cyan
Green
Yellow
Red
Magenta
White
3/Color
Black
Kodak Gray Scale
© Kodak, 2007 TM: Kodak
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19

べき乗剰余演算の依頼計算と高速化に関する研究

1996年6月

川 村 信 一

目次

図目次

表目次

第 1 章

序論

1.1 研究の背景

情報社会の健全な発展を実現するために, 通信情報ネットワークの安全性（情報セキュリティ）について盛んに研究が行われている. セキュリティは技術のみによって確保できるものではないが, 技術によってカバーできる部分は少なくない. 特に計算機科学の最近の大きな発見の一つである公開鍵暗号の概念の発表（1976 年）以来 [56], 公衆網での秘匿・認証の基本的な課題はこれによってほぼ解決されるものとの期待が高まった. 程無く RSA 暗号 [57] が発表されたが, 商用網でこれを用いるにはまだ処理時間やコストの問題, 個別の場面での安全性評価など課題は多かった. 本研究は RSA 暗号に代表される公開鍵暗号の実用化を軸に情報セキュリティの実現を目指すものである[1].

1.2 本論文の構成

RSA 暗号は十進 100 桁以上の大きい整数を法（モジュロ）とする「べき乗剰余演算」によって実現され, これを高速で行う事が大きな課題であった. 処理時間を短縮するために, 本論文では 2 つのアプローチを採る. 一つは RSA 暗号の変換に繰り返し現れる処理をあらかじめ計算しておいた関数表で置き換える方法で, 以後これをテーブル参照法と呼ぶ. テーブル参照法は日本を中心に研究され種々の方式が考案された. 本論文の第 2 章で提案するテーブル参照法の特徴は, 従来方式よりも適用対象は限定されるものの, 限定されているが故に従来方式よりも処理時間が短い点である.

RSA 暗号の高速化のもう一つのアプローチは「依頼計算」によるものである（3 章〜6 章）. 依頼計算とは, 計算力が小さい計算機（クライアント）が処理に関わる秘密を漏らす事無く, 高速な補助装置（サーバ）から計算力を借りるためのプロトコルを指す. 例えば, IC カード[2]はセキュリティ・システム構築のキーコンポーネントと考えられているが, 計算力が小さいので現状では RSA 暗号の実装には向かない. 依頼計算を適用することで IC カードが補助装置の計算力を安全に借りることができれば IC カードの実用化を早められる可能性が大きい. また仮に近い将来単独で実用的な時間内に RSA 暗号を処理で

[1] 公開鍵暗号および RSA 暗号については付録 A.1 を参照.

[2] IC カードはスマートカードとも呼ばれキャッシュカード大のプラスチックカードに CPU とメモリを封止したもので, 携帯可能な個人対応の情報処理装置として様々な応用が考えられている.

きる IC カードが登場したとしても, 同じ計算力でより高速に処理ができればさらに応用が広がろう. 本論文の依頼計算研究の特徴は安全性を零知識性やエントロピーを用いて最大限厳密に評価している点である.

第7章では, これまで検討してきた暗号要素技術を応用したシステムとして暗号電子メールシステムについて述べる. 具体的にはワークステーションを結ぶ LAN 上で動作するプロトタイプを試作した. Fiat-Shamir 法を利用した鍵配送プロトコルにも新規の工夫がある.

以上に述べた本論文の構成を図 1.1に示す.

1.3 結果の概要

この章の最後に本論文の主たる成果をまとめる.

- べき乗計算アルゴリズム：
  二進法に基づいてべき乗を剰余乗算に分解した場合, 全体の 1/3 は変数を定数倍してモジュロ演算をおこなう定数乗剰余演算である. 従来の高速化手法では法 N が定数であることのみを利用しており, 乗数も定数で有る点は看過されてきた. 本論文では乗数も定数と見なせることに着目して, 従来のべき乗計算よりも高速なテーブル参照に基づくべき乗アルゴリズムを提案する. (表 1.1参照). 提案したべき乗アルゴリズムは入力サイズが十分大きいとき, 従来方式と比べ 17% 処理時間を削減できることを理論的に導いた. また提案アルゴリズムをワークステーション向けソフトと DSP(Digital Signal Processor) 上に実装しその処理速度を測定し, 従来方式より 14% 処理時間を短縮できることを確認した.

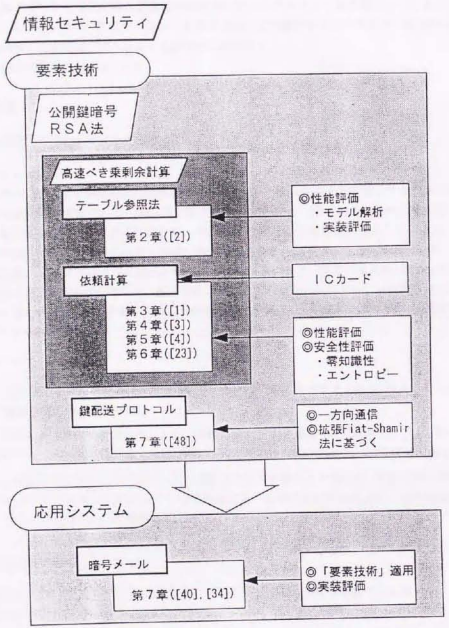
- 依頼計算について：

  1. まず、安全性は必ずしも証明付きではないが, 高速性で他の方式よりも勝る, 松本らの方式について処理速度の解析を行った.
  2. 次に, 処理速度はある程度犠牲にしても（計算結果の正誤を漏らさない限りは）, 安全性の証明付きのプロトコルを提案する. これは同じ安全性を持つ従来方式に比べ処理速度で勝る.
  3. さらに, 前記安全なプロトコルにおいて計算結果の正誤を漏らした場合に, 秘密指数に関する情報がどういうメカニズムによってどの程度漏れるかを明らかにすると共に, 漏洩情報量を少なく押さえる対策について考察した.
  4. 最後に, 計算結果の正誤を漏らしても, 安全性が確保できるような方式を提案し, その安全性を一部証明付きで示した（表 1.2）.

  実用的に意味のある問題について, 安全性を落とすことなしに処理速度を改善できる事を示したことに本研究の意義がある.

- 暗号電子メール：
  公開鍵暗号が実システムで有効に働くことを実証すると共に, ユーザの要求に応えるために, 電子メールシステムの文書の秘匿, 改ざんの検出および発信者の認証を行うプロトタイプシステムを

LAN 上で開発した. 実装に当たってはテーブル参照に基づくべき乗アルゴリズムを用いると共に, 依頼計算を用いた場合の性能評価を行なった. プロトタイプシステムは 100 名程度のユーザの下で試験運用され, 特に問題なく使う事ができることが確認できた.



注：は[]主な文献番号

図 1.1: 本論文の構成

表1.1：べき乗剰余演算の高速化（1）

[剰余乗算を高速化するテーブル参照方式]

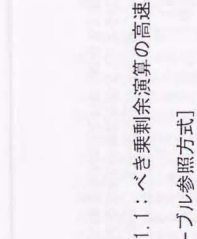
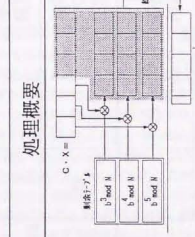
	適用対象	処理概要	備考
田中・岡本法	C・X mod N (C，Xは変数 Nは定数)		・汎用プロセッサ向き (←→専用　)
提案方式	C・M mod N (Cは変数、 M，Nは定数)		・汎用プロセッサ向き ・定数倍剰余演算に限れば田中・岡本法の2倍高速。2進法のべき乗に適用した場合、十分Nが大きければ処理時間を17%削減

表1.2：べき乗剰余演算の高速化（2）

［依頼計算］

プロトコル ＼	結果の正誤取扱い条件	耐攻撃性＊ 受動攻撃	耐攻撃性＊ 能動攻撃	速度	検討の概要
S2法（松本他）	制約有り：サーバに渡さない	○	○	◎	松本らが提案した方式について、ｸﾗｲｱﾝﾄ・ｻｰﾊﾞの速度比と処理速度のﾄﾚｰﾄﾞｵﾌ解析．通信速度も考慮（第3章）
提案方式プロトコル1～4		◎	◎	○	秘密指数に関して零知識な方式を提案．同じ安全性の従来方式より高速（第4章）
	制約無し：サーバに渡す	◎	△		正誤結果から秘密情報が漏洩するﾒｶﾆｽﾞﾑとｴﾝﾄﾛﾋﾟｰを尺度に漏洩を評価．対策も示す（第5章）
提案方式プロトコル5		◎	○	△	秘密指数を2数の積に分解して安全性を高めることを検討．能動攻撃に対する部分的安全性を証明（第6章）

＊耐攻撃性：　◎…証明突きで安全、　○…安全性の証明は無し、　△…特定の場合攻撃法有り
受動攻撃…正しいﾌﾟﾛﾄｺﾙのやりとりのみから秘密を得ようとする攻撃
能動攻撃…ﾌﾟﾛﾄｺﾙで規定されている他以外の値をｸﾗｲｱﾝﾄに与えて秘密を得ようとする攻撃

第2章

テーブル参照による高速べき乗剰余演算

2.1 はじめに

現代暗号，特に公開鍵暗号の多くは整数論の問題に基づいて構成されており，非常に大きい整数を法とするべき乗剰余計算を基本演算として構成されることが多い．この演算は多くの処理ステップを必要とし，多大な処理時間を要す．従って，暗号の応用を考える上で最も関心を集めている問題の一つは，このべき乗剰余演算の処理時間をいかにして削減するかという問題である．この章では，新しいテーブル参照法を用いた高速のべき乗剰余計算アルゴリズムを提案する．

べき乗剰余演算を行うには，通常，これを一連の剰余乗算に展開する．その際，最も広く用いられている分解法として二進法（Binary Algorithm）があり，この章でも二進法を基本アルゴリズムとして用いる．二進法は加算連鎖（Addition Chain）と呼ばれる展開アルゴリズムの一種で簡潔で使用するメモリ量が最も少なく，規則性もよいため用いる事にした．こうしてひとたびべき乗を剰余乗算の繰り返しに展開するアルゴリズムが決まれば，展開された剰余乗算をどう効率良く実現するかが残された課題となる．剰余乗算を効率的に行なうために，事前に計算でき，それ以後の処理で繰り返し用いられる値をあらかじめ計算して表（テーブル）として用意しておくテーブル参照法が幾つか提案されている [62][63][64][7][65][70]．テーブルを用いないこれ以外の高速な剰余乗算法として Barret[67] や Montgomery[68] の方法があり，DSP（Digital Signal Processor）で実装したという報告もある [67][69]．これら全ての方式で利用されている事実は，法 N は繰り返し行われる剰余乗算の全てに共通であり，定数と見なせるという点である．また，別の共通点は，いずれも基本的には乗算と剰余計算を独立な演算として扱っている点も挙げられる．しかしながら，もしここで乗算と剰余計算を同時に行う事が出来れば，これらのアルゴリズムよりも性能向上を達成できる可能性がある．新アルゴリズムはまさにこの点に着目し，開発されたものである．

先に挙げた二進法は，べき指数を最上位ビットから先に処理するか，最下位ビットから処理するかで2通りの実現方法がある．後に詳しく述べるが，本章で着目するのは，べき指数を最上位ビットか

ら先に処理した場合, 展開の結果得られる一連の剰余乗算の内, 1/3 は法 Nばかりでなく乗数 Mも定数とみなすことができる, という点である. その結果として, 多少の前処理の増加はあるものの, Mと Nに基づくいくつかの代表的な値について事前に剰余を計算しておく事によって, 全体の処理時間を短縮することが可能となる. 別の言い方をすれれば新剰余乗算アルゴリズムは Mを被乗数に掛ける処理と, 剰余をとる処理とを同時に行う方式だということができる.

本章ではまず, 第 2.2節で提案アルゴリズムを示し, 次に第 2.3節で評価のための解析モデルを導入する. 第 2.4節では計算機実験による性能を示し, 第 2.5節では結果を考察する. さらに第 2.6節では提案アルゴリズムを DSP に実装した結果について述べ, 第 2.7節で本章を結ぶ.

2.2 アルゴリズム

一般にアルゴリズムの効率はそれが実装されるハードウェアの構成に依存すると考えられるが, ここではハードウェアとして, 厳密な規定とはいえないが, 標準的なプロセッサ（汎用のマイクロ・プロセッサや DSP）を前提として考える. 標準的なプロセッサを前提にして得られた解析結果は, 専用ハードウェアを開発する場合の重要な指標にもなると期待される.

2.2.1 べき乗計算の二進法

べき乗剰余計算を効率よく一連の剰余乗算に展開する方法として広く知られている二進法を説明する. まず, Mを E乗して Nで剰余をとった結果を Cとする.

$$C = M^E \bmod N \tag{2.1}$$

ここで, C, E, Mおよび Nはそれぞれ m ビットの整数とする. RSA 暗号に用いられる場合には, 安全性の観点から m は少なくとも 512 ビットは必要であると言われている. 図 2.1はC言語に倣った疑似言語で二進法を書き表したものである. ここでは, べき指数 Eは二進数で表されていて左のビットから右のビットへと（最上位ビットから先に）処理されていて, べき乗は次の様な手順で実行される:

まず, べき指数は一ビットずつ評価されてゆくが, この時一ビット処理が進む毎に次の (a),(b) の処理を行う.

(a) 中間変数 Cを 2 乗し, 法 Nによる剰余をとった後, 中間変数 Cに再び代入する.

(b) もし, 現在処理されている Eのビットが 1 であれば, その時に限り中間変数 Cにさらに定数 Mを掛けて法 Nで剰余をとり Cに代入する.

ここに述べたように処理のループの中には (a), (b) 2 種類の剰余乗算が現れる. それぞれを次のように呼ぶことにする.

(a) 二乗剰余演算

(b) 定数乗剰余演算

もし, べき指数 Eがランダムに選ばれたとすると, Eのハミング重みは非常に高い確率で $m/2$ に近い値をとる. 従って一度のべき乗計算の中で, 平均すると二乗剰余は m 回, 定数乗剰余は $m/2$ 回実行される.

```
Modexp( M, E, N){
 /* E = e[m-1]e[m-2] ...e[0] */
 /* Assume e[m-1]=1 */
 /* mod は剰余演算を表す */
 C=1;
 for(i=m-1; i>=0; i--){
  C=(C*C) mod N; /* 二乗剰余 */
  if( e[i]==1){
   C=(C*M) mod N;   /* 定数乗剰余 */
  }
 }
 return(C);
}
```

図 2.1: 二進法によるべき乗計算

2.2.2 テーブル参照による剰余演算

一般には剰余計算は, 乗算に比べて複雑であり処理時間もかかる. 剰余乗算の処理速度を上げるために高速に除算を行なうアルゴリズムが工夫されている. その際, 処理の高速化をはかるために多くの場合考慮されている事実は, 法 Nがべき乗のなかで繰り返される剰余乗算の全てに共通であるという点である. テーブル参照による方法では法 Nによる除算を繰り返す代わりに, 代表的な剰余を予め計算しテーブルの形で記憶しておき, 除算の一部をテーブル参照で置き換えるのである. これまでに知られているテーブル参照による剰余演算アルゴリズムを表 2.1にまとめる. ここに挙げたアルゴリズムは 2 つの m ビットの整数の剰余乗算を効率化する為に提案されたものであり, 従って前節で挙げた二乗剰余演算, 定数乗剰余演算のいずれに対しても適用可能で, どちらか一方に特化した方式ではない. この章の冒頭で述べた様に, 我々は標準的なプロセッサに適したアルゴリズムの開発を目指しているが, この方針に最も叶っているのは表に挙げた方式の内, 田中・岡本によって文献 [65] で提案された方式である. このアルゴリズムはテーブルの規模が比較的小さく, しかもその作成時間も短くて済むという点で優れている. そこで以下では二乗剰余演算については田中・岡本法をそのまま採用する事にする. 著者らが次節で提案する, テーブル参照による定数乗剰余演算アルゴリズムは, 田中・岡本アルゴリズムを定数乗剰余演算用のために改良したものと位置付けられる.

田中・岡本法を説明するために, ここで幾つかの準備をしよう. 法 Nは b 進数表現したとき l桁の

表 2.1: テーブル参照法（従来方式）

	Procedure
Fujitsu[62] Hitachi[63] Matsushita[64]	Residues are prepared for arbitray bit patterns for each block.
Toshiba[7]	Residues, coresponding to an arbitrary bit pattern for a block, are recurrsively used for other blocks.
NEC[96]	Only one residue is prepared for one block.

数であるとし，Cは $2l$桁の数であるとする．すなわち，

$$b^{l-1} \leq N \leq b^l - 1 \tag{2.2}$$

$$C = \sum_{i=0}^{2l-1} C_i \cdot b^i \qquad (0 \leq C_i \leq b-1) \tag{2.3}$$

装置実装上は通常，b は 2 のべき（2^k），特に計算機の内部ワードの精度に採るのが望ましい．Zを $C \bmod N$とする．mod 演算の分配則によって次の等式が成立する．

$$Z = \left\{ \sum_{i=1}^{l-1} C_{i+i} \cdot (b^{l+i} \bmod N) + \sum_{i=0}^{l} C_i \cdot b^i \right\} \bmod N \tag{2.4}$$

田中・岡本法は

$$b^{l+i} \bmod N \qquad (i = 1, \cdots, l-1) \tag{2.5}$$

の部分を予め計算しておく事によって剰余演算の効率化をはかる．

[田中・岡本法]

ステップ 1　剰余を計算し，それらをテーブル T1 に保存する．テーブルの第 i 要素 T1[i] は次のように定義される．

$$T1[i] = b^{l+i} \bmod N \qquad (1 \leq i \leq l-1)$$

ステップ 2　$MOD(C, N, T1)$ ルーチンの呼出：

2 - 1　中間結果 Y を次の式に従って計算する．

$$Y = \sum_{i=1}^{l-1} C_{i+i} \cdot T1[i] + \sum_{i=0}^{l} C_i \cdot b^i$$

2 - 2　補正演算 $Z = Y \bmod N$によって最終結果 Zを得る．

ステップ 3　ステップ 2 を別の Cに対して繰り返す．（終）

この方法は剰余演算を乗算と累積加算で実現している．ステップ 2 - 2 の補正演算が必要となる理由は，ステップ 2 - 1 で計算された Yが通常 Nより大きくなるからである．ただし Yと Nの桁数の差は大きくないのでこの部分は効率よく計算できると考えて良い．補正演算の具体的方法として剰余テーブルの繰り返し参照や，除算が考えられる．除算を用いたとしても Yは高々$(m + b + \lfloor \log_2 l \rfloor)$ビットの数であり，補正に要する時間は比較的短いと考えられる．また別の選択支として Yに対する補正を行わず，Nより桁数が大きいままで次の乗算を行なう事も可能である．その場合には，Nより小さい数にまるめる補正計算は最終結果に対してだけ行えば良い（文献 [70] 参照）．

2.2.3　定数乗剰余演算の為のテーブル参照法

この節では被乗数に定数を乗じることと剰余演算とを同時に行うための新しいアルゴリズムを提案する．このアルゴリズムは既に述べたように定数乗剰余演算では法 Nばかりでなく乗数 Mも定数と見なせることを利用している．このアルゴリズムは定数乗剰余演算に特化しているため，田中・岡本法を定数乗剰余演算に適用した場合と比べてより短時間で処理を終えることができる．

まず，被乗数 Cは b 進数表現されているものとする．

$$C = \sum_{i=0}^{l-1} C_i \cdot b^i \qquad (0 \leq C_i \leq b-1) \tag{2.6}$$

前節では Cは $2l$桁であったが，ここでは l桁であることに注意．法演算の分配則により，Zは次のように変形される．

$$\begin{aligned} Z &= \{C \cdot M\} \bmod N \\ &= \sum_{i=0}^{l-1} C_i \cdot (M \cdot b^i \bmod N) \bmod N \end{aligned} \tag{2.7}$$

この変形に基づき，Zは次のような手順で計算できる．

[提案方式]

ステップ 1　剰余を計算し，それらをテーブル T2 に保存する．テーブルの第 i 要素 $T2[i]$ は次のように定義される．

$$T2[i] = (M \cdot b^i) \bmod N \qquad (1 \leq i \leq l-1)$$

ステップ 2　MUL&MOD（C，N，T2）ルーチンを呼び出す：

2 − 1　中間結果 Yを次のように計算する．

$$Y = \sum_{i=0}^{l-1} C_i \cdot T2[i]$$

2－2 補正演算により $Z \equiv Y \bmod N$ を満たす Z を得る.

ステップ 3 次の C についてステップ 2 を繰り返す. (終)

ステップ 1 における剰余の事前計算に要する時間は全体の繰り返し回数が大きくなれば実質的に無視することができる. なお, 前節の最後に述べた補正演算に関する注意は, 上のステップ 2 - 2 にも適用される.

2.2.4 べき乗剰余アルゴリズム

これまでに述べた 2 種類の剰余乗算アルゴリズムを用いて, 新しいべき乗剰余アルゴリズムを構成する. アルゴリズムの流れを図 2.2に示す. 二乗剰余演算部に 2.2.2節の田中・岡本法を, 定数乗剰余演算部に 2.2.3節の方式を適用している.

```
Modexp( M, E, N){
 tgen( N, q^l, T1); /* テーブル T1 生成 */
 tgen( N, M, T2); /* テーブル T2 生成 */
 C = 1;
 for( i=m-1; i>=0; i--){
  C = C*C;
  C = MOD(C, N, T1);
  if(e[i]==1){
   C = MUL&MOD( C, N, T2);
  }
 }
 return(C);
}
tgen( N, a, T){ /* テーブル生成 */
 T[0] = a;
 for( i=1; i<=l; i++){
  T[i] = T[i-1]*B mod N;
 }
}
MOD( C, N, T1){ /* T1 による C mod N の計算 */
   ( Refer to section 2.2.2 )
}
MUL&MOD( C, N, T2){ /* T2 による (C*M) mod N の計算 */
   ( Refer to section 2.2.3 )
}
```

図 2.2: 提案するべき乗剰余アルゴリズム

2.3 評価モデル

ここでは, 提案アルゴリズムの効率を評価するためのモデルを導入し, べき乗剰余演算の実現法の異なる次の 3 つのべき乗剰余アルゴリズムに関して性能比較を行う.

方式 1 ：全ての剰余乗算を普通の乗算と除算で実現

方式 2 ：全ての剰余乗算を田中・岡本法で実現

方式 3 ：前節で示した提案方式

表 2.2に 3 方式の内容をまとめる.

表 2.2: 各方式の定義サマリー

	Exponentiation	
	Mod. Square	Mod. Mul.
Method 1	Mul. + Div.	
Method 2	Mul. + TO method	
Method 3	Mul. + TO method	Proposed

Mul. = Multiplication, Div.=Division, TO=Tanaka-Okamoto

次に記号 mul, div を以下のように定義する.

mul ：2 つの m ビットの数を掛け合わせるのに要する時間.

div ：$2m$ ビットの被除数を m ビットの除数で割るのに要する処理時間.

我々は k ビット ($\ll m$) の 2 数を掛け合わせる演算を単精度乗算とする標準的なプロセッサの使用を前提としており, またビット数 m は k よりも大きいので, mul の大部分の時間は $\lceil m/k \rceil^2$ 回の単精度の乗算と累積加算によって費やされることになる. mul、div はこれからべき乗剰余演算アルゴリズムを評価するのに基本となる時間であり, また方式 1 の処理時間の大部分を占める時間である.

評価モデルを導出するに当たって, 次の 4 つの仮定を設ける：

[仮定]

1. m が充分大きければ, mul と div は m^2 のオーダの量として表すことができる.
2. 剰余テーブルを作成するのに要する時間は div に比例する. この仮定の下テーブル作成時間は c を定数として $c \cdot div$ と表すことができる.
3. MODおよびMUL&MODルーチンのステップ 2 - 1 に現れる乗算／累積加算の演算に要する時間は mul に等しい.

4．MODおよびMUL&MODルーチンのステップ 2-2 に現れる補正演算に要する時間は全処理時間に比べて無視できる．

仮定 1 ～ 3 は各処理内容に鑑み，導入した．仮定 4 は主としてモデルを単純化する目的で導入した．仮定 4 を置かない場合，補正演算に具体的に用いる事のできる演算が幾通りか存在するため，モデルは複雑となり，また汎用性も低くなる．ここに挙げた仮定の妥当性については後ほど 2.5節で検証する．

t_iで方式 i $(i = 1, 2, 3)$ の実行時間を表すものとする．すると，仮定により

$$t_1 = (3/2)m(mul + div) + o(m^3) \tag{2.8}$$

$$t_2 = 3m \cdot mul + c \cdot div + o(m^3) \tag{2.9}$$

$$t_3 = (5/2)m \cdot mul + 2c \cdot div + o(m^3) \tag{2.10}$$

ここで $o(m^3)$ は次のように定義する．

$$\lim_{m \to 0} \frac{o(m^3)}{m^3} = 0 \tag{2.11}$$

R_1, R_2および R_3を t_i間の比と定義すると次の近似が成り立つ．

$$R_1 = t_2/t_1 \approx \frac{6 + 2cr/m}{3(1 + r)} \tag{2.12}$$

$$R_2 = t_3/t_1 \approx \frac{5 + 4cr/m}{3(1 + r)} \tag{2.13}$$

$$R_3 = t_3/t_2 \approx \frac{5 + 4cr/m}{6 + 2cr/m} \tag{2.14}$$

ここで $r = div/mul$と定義する．近似式を導く際，$o(m^3)$ は無視できるものとした．なお，ここで t_iの比 R_iを導入したのは，R_iは t_iに比べて個々のマシンやプログラミング技術の影響が少ないと考えられるからである．R_iにおいては定数 c および rのみがマシンその他に依存する．

R_1と R_2の値を評価するには，rと c の値を計測しなければならない．実験によると cや rは通常 1 から 10 の範囲の値と考えてよい．一方，m が充分大きいとすれば R_3は r, c の値に依存せず，0.83 である．言い替えれば方式 3（提案方式）は方式 2 に比べて 17%処理時間を削減する．

テーブル作成に要する相対時間 R_4は次の様に表される．

$$R_4 = \frac{4cr/m}{5 + 4cr/m} \approx \frac{4cr}{5} \frac{1}{m} \tag{2.15}$$

（固定の kに対し）mが充分大きいとき，R_4は m に反比例する．従って m の増大に伴ってテーブル作成時間は無視できるようになる．

最後に，方式 iでテーブルが占めるメモリ量を W_iで表すと，

$$W_1 = 0 \quad [\text{ビット}] \tag{2.16}$$

$$W_2 = m \cdot (l - 1) \quad [\text{ビット}] \tag{2.17}$$

$$W_3 = m \cdot (2l - 1) \quad [\text{ビット}] \tag{2.18}$$

ここで lは法 Nを 2^k進数で表した時の桁数を表す．

2.4 計算機実験

表 2.2に示した 3 つのべき乗剰余演算のアルゴリズムを計算機ソフトウェアで実現し，その性能をいくつかの m について測定した．シミュレーション条件を表 2.3に示す．図 2.3は t_1, t_2, および t_3 の測定値を示しており，いずれもテーブル作成時間を含んでいる．図 2.4は R_1および R_2を百分率で示す．

表 2.3: シミュレーション条件

Machine	TOSHIBA AS3260C (CPU: 68020, CLOCK: 25MHz)
Language	C
Data Structure	1 block = 16 bit, i.e., $b = 2^16$
	m bit = $\lceil m/16 \rceil$ blocks
Measured time	From (M,E,N) input to C output
	(including table generation time)

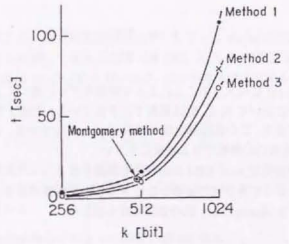


図 2.3: べき乗時間

暗号の応用で重要な $m = 512$ の場合を中心にシミュレーション結果をまとめる．方式 3 は方式 1 に比較して 37%の処理時間削減を実現しており，方式 2 に比べた場合には 14%削減している．実験に

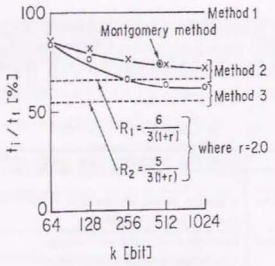


図 2.4: 正規化処理時間

より得られた rと c の値は,

$$r = 2.0 \tag{2.19}$$

$$c = 3.1 \tag{2.20}$$

これらの値を式 (2.12) − (2.14) に代入することによって評価モデルに基づく R_1, R_2, R_3および R_4 の評価値が得られる. 図 2.4において R_1と R_2は破線で示されている. 方式 3 でのテーブル作成の相対時間を表す R_4は 0.97%であり, この値は測定値 0.9%と良く一致している. また, テーブル作成時間は全処理時間に比較して実質的に無視できる程度に小さい.

以上に述べたのと同様の結果は $m = 1024$ の場合にも観測される. $l = \lceil m/16 \rceil$ によって lを計算すれば式 (2.16) − (2.18) によって各方式で必要となるメモリ量が計算できる. 図 2.3, 2.4には比較のために $m = 512$ に対応する Montgomery 法の処理時間も示した.

2.5 考察

(1) 解析においても実験においても方式 3 の処理時間見積結果は $T1$, $T2$ 両テーブル作成時間が含まれている. 従って図 2.4に示された結果は法 N, 底 M, さらにべき指数 E の全てを毎回変えながら, 連続してべき乗剰余を計算するという最も一般的な状況 (以下で定義する「状況 A」) での処理速度の見積にそのまま利用できる. 一方, 想定する状況によっては繰り返しべき乗を計

算する時に, 毎回 $T1$,$T2$ を計算し直さなくても良い場合がある. この点に着目すると, 次に挙げるような A ~ C の 3 つの状況に分けることができる.

状況 A: べき乗計算が繰り返し行われる時, N,M,Eのすべてがべき乗の都度変化する場合.

状況 B: 繰り返される複数回のべき乗計算において, Nが固定されていて, Mがその都度変化する場合.

例) RSA 暗号を用いた通信において, 特定の受信者に宛てられた複数の暗号文をその受信者が復号する場合.

状況 C: Nも Mも共に定数である場合 (べき指数 Eのみが毎回変化).

例) Diffie-Hellman 鍵配送法を利用しようという通信システムにおいて, 信頼おけるセンタが各ユーザの公開鍵を一括発行する場合.

状況 B においては, テーブル $T2$ の作成は, べき乗の回数に依存せず一度だけ行えば良く, 状況 C においては $T1$, $T2$ 両テーブルとも一度計算するだけで良い. こうした特殊な状況は, テーブル生成が比較的多くの時間を要する場合 ($c \cdot r$の値が大きい場合) にはメリットが大きい. しかしながら, 2.4節のシミュレーション結果に関する限り, テーブル作成の手間が小さいので状況 A, B, C の差は非常に小さい. 既に述べた様にテーブル作成時間は全処理時間の 1%に満たないからである.

(2) 方式 3 (提案方式) の処理時間短縮効果は主として $r = div/mul$の値に依存している. 方式 3 が方式 1 および 2 に比較して優れるための条件は, R_2と R_3が共に 1 より小さい事と等価である. このことから式 (2.13),(2.14) を用いると, 次の不等式が成立する時, 方式 3 が他の方式より優れることがわかる.

$$\frac{2}{3 - 4c/m} \leq r \leq \frac{m}{2c} \tag{2.21}$$

実験によれば c は通常 1 から 10 程度の値をとる. mが 512 以上の値に選ばれた場合, 上の不等式は非常に広い範囲に亘る rの値について満たされる. 従ってほとんどの場合, 提案方式は他の方式よりも優れる.

(3) Montgomery 法は標準的なプロセッサ向きの剰余計算法としてこれまで提案された方式の内, 最も処理時間の短い方式の一つである. この方式は剰余演算を多重精度の乗算に置き換える方式であると解釈することができる. その意味で, Montgomery 法による剰余計算は田中・岡本法による剰余計算と同程度の性能を持つであろうという粗い見積が成り立つ. またそうであるならば, これまでの考察から方式 3 によるべき乗剰余アルゴリズムは Montgomery 法を用いて構成されたべき乗アルゴリズムよりも優れているという予想が成り立つ. これは図 2.4のグラフに示された結果と一致する. さらに Montgomery 法では法 Nの値が 2^mと互いに素である事と

言う制約があるので, この条件が満たされない場合には適用できない. これに対し, 提案方式には N にこのような制約はない.

(4) ここまでの評価では, 二乗計算は普通の乗算に比べてしばしば高速に実現できるという事実は一切考慮されていなかった. しかしながら, 実際に二乗計算が乗算に比べて高速に実現された様な場合には, 提案方式は他の方式に比べさらに一層優れた特性を示す. この点を具体的にモデルに基づいて検証してみる. sqr を一度の二乗計算に要する処理時間とし, さらに s を $s = sqr/mul$ と定義する. この時 R_3 は次の様に書き換えられる.

$$R_3 = \frac{3 + 2s + 4cr/m}{4 + 2s + 2cr/m} \tag{2.22}$$

仮に s を 0.5, m が充分大きいとしよう. この時 R_3 は 0.8 となり, 方式 3 は方式 2 に比べ 20%の処理時間短縮を達成できることになる. これは先に計算した 17%と比べさらに 3%の改善である.

(5) 図 2.4 において解析モデルに基づく評価と計算機シミュレーションの差が何によるのかを考えよう. 解析モデルでは m は充分大きいものと仮定しているから, m が充分大きくない領域での差は主としてこの点に起因しているものと考えられる. そこで以下の考察では m が 512 ないしそれ以上の場合に限定する. 解析モデルは 4 つの仮定に基づいて構成されていた. シミュレーションに用いたプログラムの各部の処理時間を詳細に解析した結果, 仮定 1, 2, 3 はシミュレーションとの矛盾はほとんど無く, 主たる原因は補正演算の処理時間に関する仮定 4 にあると分かった. シミュレーションでは補正演算として通常の除算を用いており, その部分に費やされた時間は, MOD (または MOD&MUL) ルーチンの処理に要する時間の 25%であった. この点を反映させて仮定 4 を修正し, MOD に要する時間を $1.25mul$ としたところ, 解析とシミュレーションは良く一致して, 補正に要する時間が両者の差であったことが確認された.

(6) 提案したテーブル参照方式は田中・岡本法を発展させたものと見る事ができる. 冒頭に述べたように, 我々が着目したのはべき乗剰余演算に現れる剰余乗算の 1/3 は定数乗剰余演算であると言う点であった. 類似の改良は文献 [62],[63] に示されている他のテーブル参照法にも適用可能と予想される. その場合にも定数乗剰余演算の為のテーブル作成に要する時間が約 $m/2$ 回の多重精度乗算に要する処理時間 (この乗算時間がテーブルの導入により不要となる) よりも短い限りは, テーブル参照法の適用による処理時間の短縮が達成できる.

2.6 DSPによる実装

提案したべき乗剰余アルゴリズムを 16 ビットの固定小数点 DSP で実現した. 使用した DSP は単精度乗算を 25 ナノ秒で実行する処理性能を持ち, RAM は 2k ワードまで利用できる。べき乗アルゴリズムの性能を表 2.4 にまとめた. なお, 実装上の都合により被除数の 30 ビットのブロック毎に (16 ビットのブロックでは無い事に注意) 一つの剰余値をテーブルに用意した. 提案方式によると 512 ビットの法を用いた場合, 0.11 秒で 1 度のべき乗剰余演算を完了する。これを RSA 暗号の暗号化または復号と見ると 512 (ビット) /0.11 (秒) = 4.5kbps のスループットを達成する事になる. ここでは RSA 暗号の復号で利用される事がある中国剰余定理に基づく高速化の技法は用いていない事に注意.

定数 r および c はこの場合それぞれ 10.6 と 3.5 であった. この値を用いると, $R_1 = 0.18$, $R_2 = 0.16$, $R_3 = 0.86$, $R_4 = 0.057$ と計算される. これらの値によると, DSP 実装の場合の方式 2 および 3 の方式 1 に対する処理時間の短縮効果は, 2.4 節の計算機シュミレーションの場合の短縮効果に比べより大きい. これは DSP 実装での r の値が計算機シミュレーションの r に比べ約 5 倍大きいことに起因している. あるいは, DSP では (多重精度の) 除算が計算機シミュレーションの場合に比べ 5 倍複雑となっていることに起因していると言い替える事ができる. シミュレーションで用いたワークステーションの CPU の場合と違い, DSP では (単精度の) 除算命令が用意されていないことが, この違いの主な原因である.

表 2.4: DSP での実装結果

	Time [sec]	Throughput [kbps]	Table size [kWord]	Program [kWord]
Method 2	0.14	3.7	0.63	3.2
Method 3 (Proposed)	0.11	4.5	1.30	4.4

1 Word = 16 bit

2.7 むすび

この章では新しいテーブル参照法に基づくべき乗剰余計算アルゴリズムについて論じた. 新提案ではべき乗剰余を二進法でべき指数を最上位ビットから走査して剰余乗算に展開したとき, 展開された一連の剰余乗算の内, 平均で 1/3 は定数乗剰余演算であることに着目して処理時間の短縮をはかっている. また解析モデルと計算機実験によるアルゴリズムの性能評価も行った.

解析モデルに基づいて提案方式を従来のテーブル参照方式 (方式 2) と比較した場合, 法 N のビット数 m が充分大きければ提案方式は 17%の処理時間削減をはかれることが示された. また計算機シミュレーションおよび DSP での実装ではともに 14%の処理時間削減が観測された.

提案方式ではテーブル作成の手間は増えるものの, その絶対値は全処理時間に比べてわずかであり, テーブル作成時間を含めて評価しても提案方式は従来方式よりも優れている.

提案方式を DSP とワークステーションのソフトウェアとしてそれぞれ実現した. これらのライブラリは暗号を利用したより複雑なセキュリティシステムの構築の為に利用できる. 特に, 第 7 章では暗号メールシステムの構築にこれを用いている.

本章では実装の容易さのため二進法に基づくべき乗アルゴリズムに, 新たに提案した定数乗剰余テーブルを適用した結果について述べた. 他のより複雑な展開法に対しても同様の着想が適用できるか, またその効果はどの程度であるかは今後の検討課題である.

第 3 章

べき乗用依頼計算の性能解析

3.1 はじめに

IC カードは暗号を通信・情報システムで利用しようという場合, 暗号化の鍵を各個人が管理し持ち運ぶための小型装置として好適である. しかしながらその計算能力は現在のところ 8 ビット CPU が主流でさほど大きいとは言えず, RSA 暗号系のような公開鍵暗号系で必要となる複雑な処理を適当な時間内に行うには不十分である. また, 仮に IC カードが他の計算力の大きい装置に鍵を転送し, そこで暗号変換を行おうとすると, 相手装置に個人の秘密である鍵情報が漏洩する恐れが生じ, システムのセキュリティは著しく低下してしまう. さらに外部装置と IC カードの間の通信を暗号化することも可能であるが, その暗号通信が IC カードでも短時間で行えなければならないこと, 外部装置が改造されてもそこから第 3 者に鍵情報が漏れる事がないようにしなければならないことを考えると必ずしも良い対策とは言い難い. それでは外部装置と IC カードとの間で秘密情報を共有することなく, 外部装置が計算力のみを IC カードに貸すことはできないだろうか? 松本らはこのような機能を実現するプロトコルとして依頼計算プロトコルを提案した [73]. 依頼計算では計算を依頼する側をクライアント, その依頼を受ける外部装置をサーバと呼ぶ. IC カードシステムの場合には, IC カードがクライアントであり, POS 端末やセンタの大型計算機がサーバに相当する. 依頼計算の目的はクライアント単独で計算を行った場合に比べ計算時間の短縮をはかる事であるが, その際クライアントの秘密情報はサーバに漏れないようにしなければならない.

この章では松本らによって提案された S2 プロトコルおよび著者らによって提案された KS プロトコルの性能の詳細な解析をおこなう. いずれも RSA 暗号の復号変換の高速化のために提案された依頼計算プロトコルである. これら二つのプロトコルは独立して提案されたが, KS 法は S2 プロトコルの特殊な場合とみなすことができる. S2 プロトコルは非常に広いクラスのプロトコルであるが, 本章の解析以前には特定のパラメータに関する性能しか明かにされていなかった. 本章の議論の目的は, プロトコルの詳細な性能解析を行うとともに, 安全なパラメータの設定法を明らかにする事である. まず, 3.2節では S2 プロトコルの定義を示す. 3.3節では、パラメータの決定法とプロトコルの実行時間の表現法について述べる. 3.4節では解析と実験結果を示す. 3.5で本章をむすぶ.

3.2 S2 プロトコル

S2 プロトコルは文献 [75] において, RSA 暗号の署名作成（または復号）変換の高速化に適したプロトコルとして提案された. その定義をここで紹介する.

[課題]

RSA 暗号の署名の作成を行うこと. すなわち, メッセージ M の法 N の下での D 乗を計算すること:

$$S = M^D \bmod N \tag{3.1}$$

ここで, Dはクライアントの秘密指数であり, Nは公開された法で, 二つの秘密の素因数 P, Q の積である（$N = PQ$）.

ここでは、署名作成を想定しているため, 秘密に保ちたいのは指数 Dであることに注意せよ. メッセージ Mは第三者に知られてもよい.

[S2 プロトコル]

準備： クライアントは秘密指数 Dを次式を満たす 3 種類の部分情報ベクトル $\boldsymbol{F}$,$\boldsymbol{G}$,$\boldsymbol{D}$ に分割する.

$$D \equiv \sum_{i=1}^{l} f_i \cdot d_i \pmod{P-1} \tag{3.2}$$

$$D \equiv \sum_{i=1}^{l} g_i \cdot d_i \pmod{Q-1} \tag{3.3}$$

ここで$\boldsymbol{F} = [f_1, \cdots, f_l]$, $\boldsymbol{G} = [g_1, \cdots, g_l]$, および$\boldsymbol{D} = [d_1, \cdots, d_l]$ である.

ステップ 1： クライアントはサーバに M, N, および$\boldsymbol{D}$を送る.

ステップ 2： サーバは次式に従って $x_i \quad (i = 1, \cdots, l)$ を計算する.

$$x_i = M^{d_i} \bmod N$$

そして, $\boldsymbol{X} = [x_1, \cdots, x_l]$ をクライアントに返す.

ステップ 3： クライアントは Y_P, Y_Qを次式に従って計算する.

$$Y_P = \prod_{i=1}^{l} x_i^{f_i} \bmod P \tag{3.4}$$

$$Y_Q = \prod_{i=1}^{l} x_i^{g_i} \bmod Q \tag{3.5}$$

さらに, 中国剰余定理により, 次の連立方程式を満たす S を定める. S が署名結果である.

$$S \equiv Y_P \pmod{P} \tag{3.6}$$

$$S \equiv Y_Q \pmod{Q} \tag{3.7}$$

（終）

このプロトコルは $l = 2$, $d_1 = 1$ という特殊な場合として KS プロトコルを含んでいる. $\boldsymbol{F}$と$\boldsymbol{G}$ の要素が 1,0 の二値に制限されている場合には, このプロトコルはバイナリ S2 法と呼ばれ, それ以外の場合は非バイナリ S2 法と呼ばれる. バイナリ S2 法と KS 法の性能解析は文献 [28] に与えられている. ここからは非バイナリ法の解析について述べる.

$\boldsymbol{F}$と$\boldsymbol{G}$を知っているのはクライアントのみで, 一方$\boldsymbol{D}$は公開する. 後に議論するように, 公開されている$\boldsymbol{D}$から秘密指数 Dを求めることが, 計算量的に困難である様に$\boldsymbol{F}$, $\boldsymbol{G}$を設計することがこのプロトコルのポイントとなる.

3.3 安全性条件と実行時間

3.3.1 攻撃法に関する考察

前節の S2 プロトコルの仕様は, 署名結果をクライアントに渡さない形で示した. それはその動作がサーバの計算力を借りる為に不可欠では無いからである. しかし署名 Sがサーバに送られることは, 実際のスマートカードと POS 端末の間の動作を考えるとしばしば起こる事と考えられる. 従って S2 プロトコルは次の様な 3 つの動作からなるプロトコルとして扱うのが自然であろう.

動作 1 ： クライアントはサーバに「要求」を送る.

動作 2 ： サーバは計算結果をクライアントに返す.

動作 3 ： クライアントは署名をサーバに返す.

この様なプロトコルの場合には攻撃者の制約の違いから攻撃は 2 種類の攻撃法, すなわち受動攻撃と能動攻撃に分けられる.

受動攻撃とはサーバがプロトコルで規定された動作を逸脱せずに得られる情報のみを用いて秘密情報を得ようという攻撃である. RSA 暗号においては任意に与えられたメッセージ Mに対する署名 Sを知っても秘密指数 Dを推定する助けにはならないと考えられるので, S2 プロトコルにおける受動攻撃は, 動作 1 で得られる情報のみに基づいて秘密指数 Dを推定することと等価である.

受動攻撃を完全に防止する手段として, 文献 [82] に示されたように, サーバに渡す情報を Dに対して全く依存しない様に設計する方法が考えられる. この様に設計することで受動攻撃に対しては秘密情報は一切漏れなくなる. しかし文献 [82] の方式は後で比較する様に, 安全性は高いが処理時間の面で S2 法に劣り, 通信コスト（クライアント・サーバ間でやりとりされるメッセージの量）も高い.

この章ではS2プロトコルが潜在的に持つ高い能力を一定の安全性尺度の下で評価するために, 文献 [75] での議論に従い, これまでに知られている最も優れた探索法で総当たり的に Dを推定した場合に, 最悪でどれだけ試行錯誤しなければならないかその回数 $N(l,n)$ によって安全性を測る事にする (l, n は次設で定義するパラメータ). S2プロトコルにおける $N(l,n)$ の具体的形は 3.3.4節で与える. この尺度はあくまでこれまで知られている最も優れた探索法を用いるという前提で考えているので, より巧妙な探索法の発見によって, 評価尺度 $N(l,n)$ は変わりうる. しかしながら, これ以後展開する議論は新たな評価尺度 $N'(l,n)$ のもと同様に成り立つものと期待される. なによりも $N(l,n)$ を評価尺度とすることの最大の利点はそれ以外の尺度 (例えば零知識性) では得られない効率の良いプロトコルが設計できることである.

一方, 受動攻撃と並ぶもう一つの攻撃法として能動攻撃がある. S2プロトコルの場合, 能動攻撃とは動作2で正しくない値をクライアントに返し, その結果が動作3にどう反映されるかを悪用する攻撃を指す. 文献 [5] では能動攻撃により法 N の素因数分解が容易に求められる例が論じられている. ただし, その能動攻撃はクライアントが署名結果Sが正しく計算されているか否かを検査することによって容易に排除することが出来る. 署名結果の検査には署名の検査式

$$M = S^E (\mathrm{mod} N) \tag{3.8}$$

を Sが満たしているかどうかを確認すればよい. ここで Eは署名検査様のべき指数であるが, これを充分小さい値 (最小値は3) に採ることで署名検査は短時間で完了するので, 検査のステップをプロトコルに追加することはプロトコルの若干の性能劣化を招くだけである. 署名検査の導入によってプロトコルの安全性は高められるがこれによって任意の能動攻撃を排除できる訳ではないことに注意を要する.

3.3.2 S2プロトコルにおける基本パラメータと相互関係

この節ではまず依頼計算プロトコルの実行時間を表現するためのパラメータを導入し, その後それらのパラメータとセキュリティ評価量 $N(l,n)$ との相互関係を論じる.

まず, サーバの処理時間 T_Sはベクトル $\boldsymbol{D}$の要素数 lに比例する.

$$T_S = k_2 \cdot l \tag{3.9}$$

次にクライアントの行う剰余乗算の回数を nで表すと, クライアントの処理に要する時間 T_Cは nに比例する.

$$T_C = k_1 \cdot n \tag{3.10}$$

従って, パラメータ lと nはプロトコルの処理時間を評価するための基本パラメータであると考えられる. lは $\boldsymbol{D}$が公開なので, 公開の情報であるが, nは本来公開の情報ではない. しかし, 非常に狡猾な攻撃者はクライアントの処理時間を観測することによって n の値を推定するかも知れない. さらに, 以下で述べる設計方針に従えば, nは与えられた lから一意に決定される. よって, lと nは共に攻撃者が知り得る値と考えるのが妥当である. この時 Nは lと nの関数として表現出来るので, これを

$N(l,n)$ と書くことにする. 以上の考察から、セキュリティ条件を次のように定義する.

[セキュリティ条件] 秘密指数の曖昧さ (可能な候補数) を $N(l,n)$ とし, たとえ l, n が既知であっても, これが与えられた数 S_e以上に保たれる事.

なお, S2プロトコルがこの条件を満たすとき, プロトコルは $(1/S_e)$-安全であると言うことにする. 定数 S_eを与えると, セキュリティ条件を満たすパラメータ lと nは, 次の手順によって探索できる.

[パラメータ l最適化手順]

ステップ1: lを固定する.

ステップ2: 不等式,

$$N(l,n) > S_e \tag{3.11}$$

を満たす最小の n を探索する.

以下我々は現実的なセキュリティ基準として $S_e = 10^{30}$を用いる. この値はDESの鍵のビット数にみる様に, 64ビットの曖昧さが有れば, 少なくとも総当たり攻撃に対しては実質的に安全である, という事実に基づいて設定した.

こうして lと nが決定されると, プロトコルの実行時間が決まる. プロトコル実行時間を T_{S2}で表せば, 次の不等式が成り立つ.

$$\max(T_C, T_S) \le T_{S2} \le T_C + T_S \tag{3.12}$$

なお, 通信時間は簡単のためここでは無視し, 後ほど通信時間も含む解析を行う. T_{S2}が上の不等式を満たすどのような値をとるかはクライアントとサーバがどの程度処理を並列して行えるかによる. 完全な並列処理が行えれば,

$$T_{S2} = \max(T_C, T_S) \tag{3.13}$$

となるし, クライアントが処理している間はサーバが処理できないようなスケジューリングであれば,

$$T_{S2} = T_C + T_S \tag{3.14}$$

となる. 当面簡単のため T_{S2}の不等式の下限で評価を進める事にする.

3.3.3 処理時間

この節では処理時間に関し, より正確な表現を導出する. クライアントが m ビットのべき乗剰余演算をおこなうのに要する処理時間を $T^C_{exp}(m)$ で表す. ここで「m ビットのべき乗剰余演算」とは

底 a, 法 N, 指数 Eがいずれも m ビットのとき二進法を用いて $a^E \bmod N$を計算することを指す. 同様に、サーバが同じ計算を行うのに要する処理時間を $T^S_{exp}(m)$ で表す. これらを用いると, T_C,T_Sは次のように表せる.

$$\begin{aligned} T_C &= n \cdot \left(T^C_{exp}(m)/(3m/2)\right) \cdot (1/4) \cdot (3/2) \\ &= n \cdot T^C_{exp}(m)/4m \end{aligned} \tag{3.15}$$

$$T_S = l \cdot T^S_{exp}(m) \tag{3.16}$$

式の導出にあたって, 次の 3 つの仮定を置いた.

(a) べき指数を二進展開した各ビットは 1/2 の確率で 1 の値をとる.

(b) $(m/2)$ ビットの剰余乗算は, m ビットの剰余乗算の 1/4 の計算時間を要する.

(c) S2 プロトコルでは, クライアントの一度の剰余乗算は m ビットの法による剰余計算と, $(m/2)$ ビットの剰余乗算からなっている. 従ってその計算は, 通常の $(m/2)$ ビットの剰余乗算の 3/2 の時間を要する.

仮定 (a) は, 式 (3.15) で, べき乗時間 $T^C_{exp}(m)$ を, 剰余乗算時間に変換する係数 $3m/2$ に反映されており, 仮定 (b) (c) はそれぞれ係数 1/4, 3/2 に対応している. クライアントの計算には中国剰余定理を利用するために, 法 P, 法 Q でべき乗を行っているのでこの様な煩雑な変換が必要となった.

次にクライアントのべき乗計算速度を 1 とした場合の, サーバの相対的なべき乗計算速度を v であらわす. 定義より,

$$v = T^C_{exp}(m)/T^S_{exp}(m) \tag{3.17}$$

これを式 (3.16) に代入すると,

$$T_S = l \cdot T^C_{exp}(m)/v \tag{3.18}$$

さらに, クライアントが単独で署名計算を行った場合の処理時間を T^Qとし, これを単位として各処理時間を正規化する. この様にする目的は, クライアント単独で署名を作成するのに比べ, 依頼計算の採用でどれだけ計算時間を短縮できるかを相対値で示すためである. T^Qは, 署名計算が中国剰余定理を用いて行われたとすると, 通常のべき乗剰余計算の 1/4 の時間で実現できるので,

$$T^Q = T^C_{exp}(m)/4 \tag{3.19}$$

プロトコルの T^Qで正規化された実行時間を T_Nで表すと,

$$T_N = T_{S2}/T^Q \tag{3.20}$$

式 (3.15), (3.18), (3.19) を式 (3.12) に代入すると, 次の不等式が得られる.

$$\max\left(\frac{n}{m}, \frac{4l}{v}\right) \leq T_N \leq \frac{n}{m} + \frac{4l}{v} \tag{3.21}$$

T_Nの逆数 A_eはプロトコルによる処理時間の短縮効果を表すことになる.

3.3.4 $N(l,n)$ の評価

パラメータ lと n を決定するには, 関数 $N(l,n)$ を具体的に lと nで表さなければならない. 定義より n は f_iと g_i $(i = 1, \cdots, l)$ を指数とする $2l$通りのべき乗計算を行うのに必要となる乗算回数の総和である. 逆に n という有限の乗算回数を $2l$通りのべき乗計算にどう分配するかを決定すると, それに対応して f_iと g_iの値が一意に決まる. $N(l,n)$ は f_i, g_iの曖昧さを意味するから, 与えられた l, n に対して, f_iと g_iに何通りの乗算分配の仕方があるかを数え上げればそれが $N(l,n)$ である.

$2l$次元のベクトル u の集合を L で表す. u の i 番目の要素 n_i (但し $i < l$) と $(i+l)$ 番目の要素 n_{i+l}はそれぞれべき指数 f_i, g_iに割り付けられた乗算の回数とする. L を用いると, $N(l,n)$ は次のように書き下せる.

$$N(l,n) = \sum_{u \in L} \left(\prod_{i=1}^{2l} \rho(n_i) \right) / 2g \tag{3.22}$$

ここで gは $(P-1)$ と $(Q-1)$ の最大公約数であり, $\rho(x)$ は x 回の乗算で実現されるべき乗計算の場合の数である. ただし, べき乗には二進法を用いるものとする. $\rho(x)$ は次の式で評価できる.

$$\rho(x) = \sum_{i=0}^{\lfloor x \rfloor} \binom{x-1}{i} \tag{3.23}$$

ここで、$\lfloor x \rfloor$ は x を越えない最大の整数を, $\binom{n}{k}$は $n!/k!(n-k)!$を表す. $\rho(x)$ の値は次の等式によって評価できる.

$$\sum_{i=0}^{\lfloor x \rfloor} \binom{x-1}{i} = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{x+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{x+1}}{\sqrt{5}} \tag{3.24}$$

この等式が成り立つことは帰納法によって確認できる.

$\rho(x)$ を用いて $N(l,n)$ を厳密に計算するのは困難なので, その近似値,

$$\rho(x) \simeq \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^{x+1} \tag{3.25}$$

を用いる. すると,

$$\prod_{i=1}^{2l} \rho(n_i) \simeq \left(\frac{5+\sqrt{5}}{10} \right)^{2l} \cdot \left(\frac{1+\sqrt{5}}{2} \right)^{n} \tag{3.26}$$

これを式 (3.22) に代入すると, $N(l,n)$ の近似式として次式が得られる.

$$N(l,n) \simeq \left(\frac{5+\sqrt{5}}{10} \right)^{2l} \cdot \left(\frac{1+\sqrt{5}}{2} \right)^{n} \cdot {}_{2l}H_n / 2g \tag{3.27}$$

ここで${}_{2l}H_n$は多重組み合わせを表し, $\binom{2l+n-1}{n}$に等しい. 式 (3.27) の導出においては, $\sum_{i=1}^{2l} n_i = n$ であること, 集合 L の要素数は${}_{2l}H_n$である事を用いている.

$N(l,n)$ を l,n で表すことが出来たのでこれを用いて, 3.3.2節の探索手続きに従って与えられたセキュリティ条件を満たす l,n の値が求められる. なお, 同様の議論に従って, S2 プロトコルのバイナリ版の評価関数 N_B を求める事が出来, その形は近似なしに次のように与えられる.

$$N_B(l,n) = \binom{2l}{n}/2g \tag{3.28}$$

3.4 数値実験結果

3.4.1 一般的結果

図 3.1に 512 ビットの法 ($m = 512$) を用いた場合のサーバの相対計算力 (クライアントの計算力を 1 と正規化) に対するプロトコルの所要時間の特性を示す. ここでの安全性条件は 10^{30} とした. 縦軸は 1 ブロックの署名作成に要する処理時間を, クライアント単独で同じ計算を行のに要する時間で正規化した値である. ここではクライアントとサーバ間の通信に要する処理時間は無視している (あるいは, 両者を結ぶ通信回線は充分高速であると仮定していると言い換えてもよい). 回線速度を考慮した結果は次節で示す. 図 3.1で用いている略号 KS は文献 [45] で提案された方式, S2 は S2 法を意味する. 従来方式 (Conventional method) は, 文献 [85] で提案されたように中国剰余定理を用いてクライアント単独で署名計算を行った場合を指す. Quisquater 等によって文献 [82] で提案された依頼計算の特性もグラフに示した (付録B.3参照). QS 法は安全性レベルを $N(l,n)$ では測れないことに注意. 図 3.1は縦軸, 横軸ともに正規化されているので, 特定のクライアントやサーバの能力に依存しない一般的な結果を示している. また, 特定のクライアントが与えられてその計算力が決まれば, グラフの縦軸を相対値から絶対値によみかえることは容易である.

表 3.1: 10^{-30}安全でのパラメータ

v	l	n
4 – 32	1	137
32 – 63	2	108
63 – 91	3	98
91 – 126	4	89
126 – 166	5	81
347 – 417	10	57
1050	20	38
4096	50	25
10779	100	19
43886	300	14

バイナリ S2 法と非バイナリ S2 法の特性は, vが 10^4を越える領域ではほとんど差がない. これは

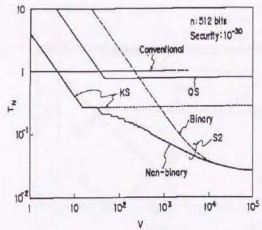


図 3.1: 正規化実行時間

次のような理由によるものと考えられる: このように vが大きい領域では S2 法のパラメータ lは n に比べてかなり大きい値に選ばれる. そのため, f_iや g_iに 0,1 と言った小さい値が割り付けられる可能性が大きくなる. その結果, 非バイナリ法の特性はバイナリ法の特性に近づくのである.

この節では非バイナリ法が, $l > 2$ で $10^2 < v < 10^4$の領域において他の方式より優れていることを示した. パラメータ l, n は安全性パラメータ S_eのみに依存して選ばれる. 従って, 他の m の値に対する特性も, S_eが同じである限りは図 3.1の特性と良く似た傾向を示すと予想される. ただし, 式 (3.21) によれば, プロトコルによる処理時間の短縮効果は m とともに大きくなることがわかる.

3.4.2 通信コスト

これまでは 通信コストは無視してきたが, 通信速度がクライアントの処理速度に比較して相対的に低い場合や, クライアント・サーバ間で通信しなければならないメッセージの量が多ければこれを無視する訳には行かなくなる. ここでは通信コストについて考察する.

クライアント・サーバ間で通信しなければならないメッセージの量を wとすると,

$$w = 2(l+1) \quad [\text{ブロック}] \tag{3.29}$$

である. ここで, wの単位は 512 ビットのメッセージを 1 ブロックと定める. これまでに述べたパラメータ設計手続きに従うと, S_eが与えられたとき, サーバの相対速度 vが決まれば最適な lも定まる. これにより通信量 wも決定される. 図 3.3は与えられた vに対して必要な通信コストを図 3.1に示した

各方式について示したものである. 縦軸は伝送ブロック数を示しており, 通信速度が定まれば容易に通信時間によみかえる事ができる.

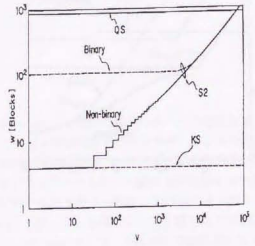


図 3.2: 通信コスト

3.4.3 現実的なクライアントに対する結果

これまでは一般的解析結果をみてきたが, ここでは現実的なプロセッサと通信速度を具体的に定めたときに, S 2法を用いることによってどの程度処理時間を短縮できるのか考察する.

まずクライアントの性能を与えよう. ここでは 8 ビットの CPU の使用を想定して, 512 ビットの法の下で中国剰余定理を用いた署名計算のスループットを 13bps であるとしよう. 処理時間に換算すると, $512/13 \simeq 39$ 秒となる. これは著者らが 8 ビット CPU で測定した処理時間を基に設定した値である. 通信回線の伝送速度は 2 通りの場合を想定し, 9.6kbps (図 3.3) と 64kbps (図 3.4) について解析した. なお, 現在製造されている多くの IC カードは 9.6kbps の通信ポートを持っている. いずれのグラフに於いても, 破線はクライアント, サーバの処理時間および通信時間の和を示している. $\max(T_C, T_S)$ と $T_C + T_S$ の差はグラフ上小さいから, 式 (3.12) は比較的タイトな上・下限を与えることがわかる.

クライアントの性能と通信速度を決定することで処理時間の特性が得られたが, 最後にサーバの計算能力を決めることでプロトコルの所要時間が定まる. IC カードシステムで容易に実現可能なサーバとして次のような 2 つのサーバを選んだ.

(A) 32 ビット CPU (クロック 20MHz)

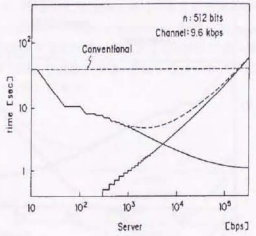


図 3.3: 現実的なクライアントに対する処理時間 (1)

(B) 16 ビット DSP (クロック 40MHz)

(B) は第 2 章で示した DSP によるべき乗計算装置である. この 2 種類のサーバと 9.6kbps の通信回線を用いた場合のプロトコルの性能を表 3.2にまとめた. サーバ A に対する結果は実際にシステムを組んで実験をし, 解析によって得られた表 3.2の結果と良く一致した. これらの例では, 通信のオーバヘッドがプロトコルの性能を非常に制限する要因となっている. クライアントの性能が比較的高く, それに比べ通信速度がプロトコルの利点を引き出す程には高くないと見る事が出来る. サーバ B の相対速度は $v = 1250$ であり, これを図 3.3に照らして考えると, これより性能が低ければプロトコルの所要時間が伸び, これより性能が高くても所要時間が短縮できないと言う意味で, 与えられた条件の下でほぼ最適なサーバである事がわかる. クライアント単独では 40 秒程度かかっていた処理をプロトコルの適用によって, サーバ A では 10.5 秒, サーバ B では 5 秒にまで短縮することができた. ここまで計算時間を短縮したことによって, 従来は処理時間の面で適用をあきらめざるを得なかった署名を利用する様々なアプリケーションへの適用大きく広がったと言える.

署名計算時間をどこまで引き下げれば良いかは, ユーザの待ち時間に対する心理までも考慮せねばならず, 署名を利用しようというシステムが提供するマン・マシンインタフェースや署名作成以外の手続きとの組み合わせ方の工夫とも関わる問題であり, ここでは詳細に論じるための材料が不足している. ただ, 署名計算の完了だけをユーザが端末の前で待つ状況を想定した時, ほとんどのユーザが待ち時間に対する不満を持たない処理時間として, 例えば 2 秒以内という値を設定して, それを実現するために必要な依頼計算のパラメータを次に考察してみよう.

まず, クライアントが先に挙げた 8 ビット CPU のままで, サーバの計算能力と通信速度は任意に

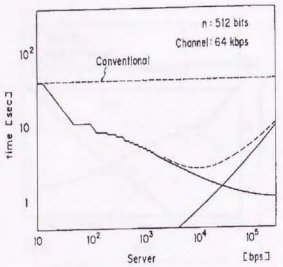


図 3.4: 現実的なクライアントに対する処理時間（2）

選択できるものとして, それらをいくらに選べば 2 秒以内で署名計算が完了するか考える. 例えば, サーバのスループットが 200kbps（512 ビットの法を用いたべき乗剰余計算を 2.56 ミリ秒で終了する処理速度）, 通信速度が 500kbps であれば, 処理時間・通信時間が各々約 1 秒でその和が 2 秒となる. ただし, IC カード・システムに限定した場合, このような条件を達成し, それが商用となるのは難しそうである.

次に, 通信速度を 9.6kbps に固定した場合に 2 秒以内の処理時間を達成するクライアントとサーバの処理能力を考える. このとき, まずクライアントは単独で RSA 署名を中国剰余定理を利用して作成した場合のスループットが 64bps の能力を持つ必要があり, サーバはべき乗剰余計算を 4.6kbps で行うことが条件となる（いずれも法は 512 ビット）. この様なクライアントは 16 ビット CPU でほぼ実

表 3.2: 実用的な例

Sever	Client (8 bit CPU; 5MHz)				
	v	l	n	Time[sec]	A_e
A) 32-bit CPU(20MHz)	16	1	137	10.5	4
B) DSP (40MHz)	1250	20	38	5	8

回線速度： 9.6 kbps
クライアント単独では 40 秒掛かる.

現でき, またサーバは上記 DSP で実現したサーバ B に相当する能力である. まとめると, S2 法を利用した場合スマートカードの CPU が 16 ビットに置き換われば現在利用できる通信速度とサーバの能力を用いて, 多くのアプリケーションに支障無く利用できる処理速度が達成できることがわかった.

最後に, 2^{-512}安全性という条件の下で描いた処理時間の特性を図 3.5 に示す. サーバの能力 v が大きくなるに従って処理時間の減少が見られるが図 3.3, 3.4 の特性に比べ削減の効果は大きくない. 同じ v に対して通信コストが前の例より大きいのは, 通信するメッセージ数を増大する事によって秘密指数の曖昧さを大きくしているからである.

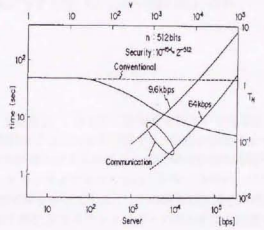


図 3.5: 2^{-512}安全の場合の処理時間

3.5 むすび

この節では S2 プロトコルの性能を詳細に解析した. S2 プロトコルは RSA 暗号の署名作成を高速化するために提案された依頼計算プロトコルの一つである. この節ではまず, 非バイナリ S2 プロトコルのパラメータの決め方を考察した. また, 計算機によりパラメータの探索を行った. プロトコルの実行時間をサーバの相対処理時間の関数としてグラフにまとめた. 特性グラフは正規化されているので, 任意のクライアント・サーバに対する処理時間をグラフから読み取る事ができる. さらに通信のコストも考察した. 最後に現実的なクライアントを想定し, 種々のサーバに対する処理時間を解析結果を基に導いた. また 8 ビット CPU をクライアントとする実験装置を作成し, 解析結果を検証した. クライアント単独では 40 秒かかる署名計算が, 実験によって 32 ビット CPU をサーバに用いることによって 10.5 秒で署名計算を完了し解析と良く一致した. また 16 ビット DSP をサーバに置き

換えると, これを 5 秒に短縮できることがわかった. 以上の結果は依頼計算が実システムで有効に働く事を示している.

第 4 章

安全性の証明付き依頼計算

4.1 はじめに

この章では安全性をより重視し, しかも同じ安全性を実現する従来方式よりも計算時間が短い, べき乗剰余計算のための依頼計算プロトコルを提案する. 前章の場合に比べてより厳密に安全性の評価を行うために, ここでは依頼計算プロトコルを 2 つの確率的多項式時間チューリング・マシンの間のプロトコルと定義する. 一方がクライアントであり, もう一方がサーバである. クライアントは計算したい関数を指定する秘密情報を持っているが, その関数を所望の時間以内に計算できないとする. そこで計算時間を短縮するためにクライアントはサーバから計算力を借りようとするが, クライアントの秘密情報はサーバに漏れないようにしたい.

依頼計算プロトコルは松本らによって最初に検討され [73],[75], その後他の研究者によっても検討が進められている. Feigenbaum らは Instance Hiding プロトコルと呼ばれる類似の二者間のプロトコルを研究しているが [77]–[80], 彼女らのプロトコルでは補助するマシンは多項式時間以上の能力を持っているという点で依頼計算とは問題設定が全く異なっている. さらに彼女らの研究では計算したい関数が多項式時間マシンでは計算困難な場合に興味がある. 従って依頼計算と Instance Hiding プロトコルは理論的にも実用的にも全く異なるものである.

べき乗計算のための依頼計算として既にいくつかのプロトコルが提案されているが [73],[75],[81],[82], それらの内で松本らによる S2 プロトコル [75] と Laih らによる S2 プロトコルの改良版 [81] が最も計算時間が短い. S2 プロトコルの詳細な解析は前章で示した通りである. しかし, これらのプロトコルの安全性は, 従来困難と考えられている一般的問題に帰着するなどといった厳密な形では示されておらずその点に今後の研究の余地があった. なによりクライアントとサーバの間で取り交わされるメッセージが秘密のべき指数と完全には独立で無く, 何らかの情報が漏れていると考えられる.

Quisquater らによって文献 [82] で提案されたプロトコルは, クライアント, サーバ間でやり取りされるメッセージが秘密指数とは独立となる様に設計されており, 松本らのプロトコルが持っている欠点を解消したものとして注目される. このプロトコルはこれまで提案されたべき乗の依頼計算プロトコルの内, 最も安全な方式である. ところが, このプロトコルには Yao 法 [60] や Knuth 法 [58] の

ような単純な高速べき乗計算アルゴリズムの効率から考えると, さほど効率よくない（処理時間の短縮化効果が小さい）という欠点を有していた.

そこでこの章では, まず秘密指数に関する知識を一切漏らさず, しかも従来の同じ安全性を実現する文献 [82] のプロトコルよりも効率の良い方式を提案する. 提案プロトコルは, 高速べき乗計算アルゴリズム Yao 法や Knuth 法に基づいて構成される. Yao 法や Knuth 法は二進法の高次への拡張と見なせるので, この節で提案するプロトコル 1 および 2 は [73] で提案された二進法に基づくプロトコルの高次への拡張とみることもできよう. この章では安全性を証明付きで示すために, 文献 [83] で導入された零知識の概念を用いる.

以下、次節ではまず 4 つのプロトコル構成を示す. 4.3節ではべき指数の安全性について検討する. 4.4節ではプロトコルの実行時間を解析する. 4.5節では結果の検算, アタックへの対策, いっそうの高速化をはかる方策等を考察する. 最後の 4.6節はむすびである.

4.2 べき乗剰余計算のための依頼計算

ここでは 4 つの依頼計算プロトコルを提案するが, いずれもべき乗剰余計算, すなわち

$$S = M^D \bmod N \tag{4.1}$$

を高速に計算する目的で構成されている. ただし, Dはクライアントの秘密情報であり, Mや Nは公開情報とする. この計算は RSA 署名の計算や [57], Diffie-Hellman 鍵配送法 [56] で典型的に現れる. RSA の場合には通常, 署名確認用べき指数 Eが公開情報に含まれる.

安全な依頼計算プロトコルを設計するためにとった基本方針は, まず効率の良いべき乗計算アルゴリズムを用意し, それを秘密指数 Dに依存する部分と秘密指数に依存しない部分の 2 つに分割し, 秘密に依存しない部分をサーバが計算するように構成することである. 採用した高速べき乗アルゴリズムは Yao 法 [60] と Knuth 法 [58] である.

4.2.1 Yao 法に基づく構成

まず, Yao 法そのものを説明しよう. ここでは法 Nのビット数を m で表し, べき指数 Dもほぼ m ビットであるとする. kはアルゴリズムで Dを走査する際の窓のサイズで単位はビットとする. さらに基底 b を $b = 2^k$と定義する. 以上のパラメータを用いて Dは次のように表現されるものとする.

$$D = \sum_{i=0}^{l-1} d_i \cdot b^i \tag{4.2}$$

ここで, $0 \le d_i \le b-1$, $l = \lceil m/k \rceil$.

さらに新たに集合 $S(j)$ を,

$$S(j) = \{i | d_i = j\} \tag{4.3}$$

と定義すると, Dは次のように書き直せる.

$$D = \sum_{j=1}^{b-1} \left\{ j \cdot \left(\sum_{i \in S(j)} b^i \right) \right\} \tag{4.4}$$

これによって次の関係が容易に導ける.

$$\begin{aligned} M^D \bmod N &= \prod_{j=1}^{b-1} \left(\prod_{i \in S(i)} M^{b^i} \right)^j \bmod N \\ &= \prod_{j=1}^{b-1} z(j)^j \bmod N \end{aligned} \tag{4.5}$$

ここで

$$z(j) = \prod_{i \in S(j)} M^{b^i} \bmod N \tag{4.6}$$

以上の準備によって, Yao 法は次のように書ける:

まず, M^{b^i}を $i = 1, 2, \cdots, l-1$ について計算する. 次に $z(j)$ を $j = 1, 2, \cdots, b-1$ について計算する. 最後に$\prod_{j=1}^{b-1} z(j)^j \bmod N$計算すれば M^Dが求められる. なお, 最後の計算は $2b-1$ 回の乗算で効率よく行える. 詳細はプロトコル 1 を参照.

Yao 法をクライアントで処理される部分, サーバで処理される部分の二つに分割して得られたのが次のプロトコル 1 である.

[プロトコル 1]

ステップ 1: クライアントは M, Nをサーバに送る.

ステップ 2: サーバは $y_i = M^{b^i} \bmod N$を次の様に計算する.

ステップ 2 - 1: 初期化.

$$y_0 \leftarrow M$$

ステップ 2 - 2: $i = 1, 2, \cdots, l-1$ について,

$$y_i \leftarrow y_{i-1}^b \bmod N$$

ステップ 2 - 3: $y_1, y_2, \cdots, y_{l-1}$をクライアントに送る.

ステップ 3: クライアントは $z(d_i)$ に y_iを累積的に乗ずる.

ステップ 3 - 1: メモリ $z(j)$ の初期化.

$$z(j) \leftarrow 1 \quad (j = 1, 2, \cdots, b-1)$$

ステップ3-2: $i=1,2,\cdots,l-1$ について y_i を $z(j)$ に掛ける.

$$z(d_i) \leftarrow z(d_i)\cdot y_i \bmod N$$

ここで $y_0=M$. $d_i=0$ であれば, その i についてステップ3-2は省略する.

ステップ4: クライアントは $\prod_{j=1}^{b-1} z(j)^j \bmod N$ を次のように計算する.

ステップ4-1: $j=b-2,b-3,\cdots,2,1$ について,

$$z(j) \leftarrow z(j)\cdot z(j+1) \bmod N$$

ステップ4-2: ステップ4-1を繰り返す.

ステップ4-3: メモリ $z(1)$ に結果 S が得られている.

(終)

ステップ2をクライアント自身が行えばプロトコル1はYaoのべき乗アルゴリズムと一致するので, プロトコル1で正しく S が計算できることは容易に納得される.

窓サイズ k を1に限定するとステップ4は必要なくなり, プロトコル1は文献[73]で提案されたプロトコルと一致する. ただし, 文献[73]のプロトコルでは計算速度を高めるために, d のハミング重みを小さくとることが推奨されているが, これは安全性を弱める結果となっている. プロトコル1ではパラメータ k に自由度が有るので, 安全性を弱めることなく計算速度の改善をはかることができる.

4.2.2 Knuth法に基づく構成

Knuthの小窓アルゴリズム (Small window algorithm) も高速べき乗アルゴリズムの一つである. プロトコル2はこのKnuth法に基づいて構成される. 前節同様 D と N は m ビットの数とする. Yao法とKnuth法の違いはべき指数 D の走査の仕方にある. Knuth法では D は最下位ビットから最上位ビットに向けて走査され, 走査中に最初に1が現れたらそのビット位置を $w(0)$ とすると共にそのビットを含むその後 k ビットを最初のブロック u_0 とする. 次の走査は u_0 に隣接するビットから再開し, 以下同様にビット1が現れる度に新たなブロックとブロック位置を定義する. その結果, 指数 D は次のように表現される.

$$D=\sum_{i=0}^{l-1} u_i\cdot 2^{w(i)} \tag{4.7}$$

ここで $1\le u_i\le b-1$ であり u_i は奇数である. このべき指数の表現を用いる依頼計算プロトコル2は次のような手順である.

[プロトコル2]

ステップ1: クライアントは M と N をサーバに送る.

ステップ2: サーバは以下のステップを行う.

ステップ2-1: 初期化

$$y_0 \leftarrow M$$

ステップ2-2: $i=1,2,\cdots,m-1$ について

$$y_i \leftarrow y_{i-1}^2 \bmod N$$

ステップ2-3: クライアントに $y_1,y_2,\cdots,y_{m-1}$ を送る.

ステップ3: クライアントは y_i を $z(j)$ に次のように累積する.

ステップ3-1: メモリの初期化.

$$z(j) \leftarrow 1$$

ステップ3-2: $i=0,1,2,\cdots,l-1$ について, $y_{w(i)}$ を $z(j)$ に掛ける.

$$z((u_i+1)/2) \leftarrow z((u_i+1)/2)\cdot y_{w(i)} \bmod N$$

ここで, y_0 としては M を用いる.

ステップ4: クライアントは $\prod_{j=1}^{b/2} z(j)^{(2j-1)} \bmod N$ を次のようなステップで計算する.

ステップ4-1: $j=b/2-1,b/2-2,\cdots,2,1$ について,

$$z(j) \leftarrow z(j)\cdot z(j+1) \bmod N$$

ステップ4-2: $z(1)$ の値をメモリ h に保存する.

ステップ4-3: ステップ4-1を繰り返す.

ステップ4-4: 最終結果 S を次の様に計算する.

$$S \leftarrow h\cdot z(1)^2 \bmod N$$

(終)

クライアントとサーバが一体である場合を考えれば, プロトコル2で正しいべき乗結果が得られることは容易に納得される.

プロトコル1と2の主な違いをみる. まず, プロトコル2のステップ2でサーバは, 必ずしもそのすべての値が使われるとは限らないのに $(m-1)$ 個の y_i の値をクライアントに送らなければならない. これに対してプロトコル1のサーバは $\{\lceil m/k \rceil\}$ 個の値をクライアントに送るだけで良い. この点はプロトコル2が不利になる. 次の違いとして, kが固定するとプロトコル2で使われるメモリ $z(i)$ の数はプロトコル1に比べて半分で済む. 更にプロトコル2のステップ4における乗算回数はプロトコル1の場合に比べやはり半分である. これらの点はプロトコル2に有利に働く. プロトコル3, 4も含めた様々なパラメータに対する性能の比較は4.4節で行う.

4.2.3 クライアント乗算回数の削減

プロトコル1と2からそれぞれの変形版としてプロトコル3と4を導く. いずれの変形版もプロトコル開始前にクライアントが事前計算をしなければならないが, プロトコル開始後にクライアントが行わねばならない乗算回数はプロトコル1, 2に比べて減っている. プロトコル1と2ではステップ4で $z(j)$ の値に乗算をするのはクライアントだったが, プロトコル3と4ではその乗算はサーバが行う様に設計されている. その結果クライアントは $z(j)$ の値をサーバに送らなければならないが, 一般にはそれによって秘密指数が漏れる可能性が生ずる. それ故, プロトコル3と4の設計の要点は $z(j)$ の値が秘密指数の情報を漏らさない様にすることである. それを可能にするために変形版では $z(j)$ の本当の値をサーバから隠すために乱数を使用する.

[プロトコル3]

準備: クライアントの事前計算.

ステップ0-1: 乱数 $r_j \in Z_N^*$ $(j = 1, 2, \cdots, b-1)$ を生成する.

ステップ0-2: 次式で定義される Rの値を計算する.

$$R = \prod_{j=1}^{b-1} r_j^j \bmod N$$

ステップ0-3: 次式を満たす R^{-1}を計算する.

$$R \cdot R^{-1} \bmod n = 1$$

ステップ1: クライアントは Mと Nををサーバに送る.

ステップ2: サーバはプロトコル1のステップ2と同じ手順で y_iを計算し, クライアントに送り返す.

ステップ3: $z(j)$ を乱数 r_jで初期化し, プロトコル1のステップ3-2と同じ手順で $z(j)$ の値を更新して行く. その後クライアントは $z(j)$ $(j = 1, 2, \cdots, b-1)$ の値をサーバに送る.

ステップ4: サーバは下記の式で定義される wの値を, プロトコル1のステップ4のクライアントの手順と同じ手順で計算し, wをクライアントに送り返す.

$$w \leftarrow \prod_{j=1}^{b-1} z(j)^j \bmod N$$

ステップ5: クライアントは最終結果 Sを次のように計算する.

$$S \leftarrow w \cdot R^{-1} \bmod N$$

(終)

$z(j)$ の初期値は準備のステップで定めた乱数であることに注意. もし初期値がプロトコル1や2の場合の様に1だったとすると, サーバに送られる $z(j)$ の値は秘密指数dに関する手がかりを漏らす可能性がある. ステップ5における R^{-1}による乗算は乱数の影響を取り除く処理である. 同様の変更をプロトコル2に施すことによってプロトコル4が導かれる. なお, 具体的手順はもはや明かと思われるのでプロトコル4の詳細なステップは割愛する.

4.3 秘密指数の安全性

プロトコル1と2は明らかに Dの秘密を漏らしていない. その理由は通信されるメッセージ M, N, y_iは全て指数 Dと独立だからである. あるいは, 同じ M, Nに対してプロトコルを2度実行した場合, たとえ異なる Dが用いられたとしても通信回線を流れる情報は同じとなることからもこれは明かである.

プロトコル3と4については, 秘密指数 Dに関して零知識 (Zero-Knowledge) であることを示せる. すなわちサーバ Sがプロトコルに指定されている動作からどのように逸脱してみても, クライアントから Dに関する新たな知識を引き出すことができないことが示せる. これを証明するために, 提案したプロトコルを, クライアント Cとサーバ Sという2つの確率的多項式時間対話型チューリング・マシン (Probabilistic polynomial time interactive Turing machine) の間のプロトコルとしてモデル化する. Sと Cへの共通入力 (Common input) x は (M, N, E) であり, 問題のサイズは Nのビット数で表す. Cへの外部入力 (Auxiliary input) はべき指数 Dである. 共通入力 xと Cの外部入力とは一定の関係 (Relation) Rを満たす. 入力 xに対してクライアント Cとの間でプロトコルを実行したサーバ S'のビュウ (View) を $S'_{C(x,D)}(x)$ によって表す. ビュウとは直感的には, サーバから見える情報を指し, サーバ自身の乱数テープと通信テープを含んでいる. 以上の問題設定は, プロトコルの達成しようとしている目的の違いを除けば, 文献 [91] の知識に関する零知識証明 (Zero-knowledge proof system of knowledge) と同じと考えて良い.

定義 1 対話型チューリング・マシンの対 (CS) は次の様なシミュレータ Mが存在するとき, 関係 Rについて零知識であるという: Mは平均多項式時間で処理を完了し, 任意の確率的多項式時間 Sと任意の $(x, D) \in R$について, 二つのアンサンブル $S'_{C(x,D)}(x)$ と $M(x; S')$ は多項式時間識別不可能 (*Polynomially indistingishable*) である. Mは S'をサブルーチンとして使うことができる.

多項式時間識別不可能性については文献 [83] の定義を参照.

定理 1 プロトコル 3 と 4 を RSA 署名作成に使った場合, 指数 D に関する新たな知識を一切サーバに漏らさない (零知識である).

証明 (スケッチ): 証明の要点はアンサンブル

$$S'_{C(x,d)}(x) = (r; x; y_1, y_2, \cdots; z_1, z_2, \cdots; w)$$

と同じアンサンブルを出力するシミュレータを構成する事である. ここで rは S'の乱数テープの内容を表す.

rは一様乱数, y_iと wは S'をサブルーチンとして呼ぶことでシミュレートできるので, シミュレータは系列 $z(j)$ の確率分布をシミュレート出来さえすればよい.

まず, y_iが全て法 Nと互いに素である場合を考える. この時, クライアントが乱数 r_jを Z_N^*から一様に選んでいるので, $z(j)$ は Z_N^*の中を一様ランダムに分布する. 従って, シミュレータも単純に $z(j)$ を Z_N^*から一様ランダムに選ぶことによってプロトコルと同じアンサンブルを出力できる. ここでシミュレータは Nの素因数を知らないので Z_N^*から一様ランダムに要素を取り出すには多少工夫が必要である. シミュレータはまず, Z_Nから一様ランダムに要素を取り出し, それが Nと互いに素である事を確認してから用いるようにする. 取り出した要素が Nと互いに素でない時にはそれを捨てて別の要素をあらためて選ぶようにすれば多項式時間で完全に発生する系列をシミュレートできる. なお, 2 数が互いに素であることを判定するアルゴリズムは多項式時間で停止することに注意.

次に, y_iに法 Nと互いに素で無いものがある場合を考える. この場合, シミュレータは法 Nとその y_iの共通因数を求めることで法 Nの素因数分解を多項式時間で行える. さらにシミュレータは公開の指数 Eから Dを導くことができ, それを用いて $S'_{C(x,D)}(x)$ を多項式時間でシミュレートできる.

従って, 任意のサーバ S'に対して, 多項式時間シミュレータを構成できる.**(証明終)**

これと同様にして素数を法とする場合のプロトコル 3, 4 も安全であることが示せる.

4.4 性能解析

4.4.1 計算量、通信量およメモリ量

提案したプロトコルを比較するために計算量, 通信量, メモリ量を評価する. ここではプロトコル 1 を例に採りながらそれらをどう見積もったかを示す. 4 つのプロトコルに関する結果は表 4.1 にまとめられている.

クライアントの乗算 :剰余乗算がクライアントの処理の大部分を占めるものとし, 他の処理に要する時間は無視できるものとする. ステップ 3-2, 4-1, 4-2での剰余乗算の回数は高々 $(2^{k+1} + \lceil m/k \rceil - 5)$ であり, 平均では $\{2^k + (1 - 1/2^k)\lceil m/k \rceil - 3\}$. 平均値を求めるにはステップ 3-2 の乗算は複数ある $z(j)$ に対して一様に行われるものとした.

サーバの乗算 :ステップ 2-2 の剰余乗算がサーバの計算で最も時間の掛かる部分である. その回数は $k(\lceil m/k \rceil - 1)$ である.

通信量 :クライアント・サーバ間の通信量は m ビットのブロックを単位として評価する. ステップ 1 とステップ 2-3 で伝送されるメッセージ数は $(\lceil m/k \rceil + 1)$ である.

メモリ量 :m ビットのブロックを単位として考えると, クライアントは $D, M, N, y_i, z(j)$ $(j = 1, 2, \cdots, 2^k - 1)$ を格納するために $(2^k + 3)$ ワードのメモリを使用する. サーバ y_iと Nを格納するために 2 ワードのメモリを使用する.

表 4.1: 計算量の比較

Protocol	Mod.Mul. (Upper: Max., Lower: Ave.)		Communication (N_2)	Memory	
	Client (N_0)	Server (N_1)		Client	Server
1	$2^{k+1}+\lceil m/k \rceil-5$	$k(\lceil m/k \rceil-1)$	$\lceil m/k \rceil+1$	2^k+3	2
	$2^k+(1-1/2^k)\lceil m/k \rceil-3$				
2	$2^k+\lceil m/k \rceil-1$	m-1	m+1	$2^{k-1}+4$	2
	$2^{k-1}+\lceil m/(k+1) \rceil-1$				
3	†(PC-1)+$\lceil m/k \rceil$-1	$2^{k+1}+k(\lceil m/k \rceil-1)-4$	$2^k+\lceil m/k \rceil+1$	$2^{k+1}+3$	2^k
	†(PC-1)+(1-1/2^k)$\lceil m/k \rceil$				
4	‡(PC-2)+$\lceil m/k \rceil$+1	2^k+m-1	$2^{k-1}+m+2$	2^k+5	$2^{k-1}+2$
	‡(PC-2)+$\lceil m/(k+1) \rceil$+1				

† (PC-1)=(RNG: $2^k - 1$ times) + (MM: $2^{k+1} - 4$ times) + (Inv: 1 time)
‡ (PC-2)=(RNG: 2^{k-1} times) + (MM:2^k times) + (Inv: 1 time)
PC: Pre-Computation, RNG: Random number generation,
MM: Modular multiplication, Inv: Inversion.

4.4.2 規格化された処理時間の見積

プロトコルの処理時間の見積を行う. 解析を容易にするために処理時間は最悪値で評価することにし, クライアント, サーバ, 通信回線は並列には動作しないとする. さらに通信回線もクライアントやサーバと同様に入力データ量に比例した時間を処理に費やすので, 通信回線をクライアントやサー

バと区別することなく扱う．なお，メモリは充分にあるものとし，処理時間だけを方式比較の評価量とする．もし，メモリコストも評価に含めるならば，処理時間とメモリコストを関係づけるための変換尺度の導入が必要となろう．

次にいくつかの記号を定義する．

T :プロトコルの全処理時間．

T_0 :クライアントの剰余乗算に費やされる時間．

T_1 :サーバの剰余乗算に費やされる時間．

T_2 :メッセージ 1 ブロックの伝送に費やされる時間．

N_0 :クライアントの剰余乗算回数．

N_1 :サーバの剰余乗算回数．

N_2 :伝送されるメッセージブロック数．

R_1 :クライアントの処理能力に対するサーバの相対的な処理能力．$R_1 = T_0/T_1$である．

R_2 :クライアントの処理能力に対する通信回線の相対的な処理能力．$R_2 = T_0/T_2$である．

記号の定義から次の等式が成り立つ．

$$T = N_0 \cdot T_0 + N_1 \cdot T_1 + N_2 \cdot T_2 \tag{4.8}$$

TをT_0で正規化した値 Fを評価量として用いる．

$$F = T/T_0 = N_0 + N_1/R_1 + N_2/R_2 \tag{4.9}$$

次に Fを最小化する手続きについて考察する．まず，N_0, N_1, N_2の具体的な表現は前節で表 4.1にまとめられており，いずれも法のビット数 mと窓のサイズ kの関数である．一旦法 Nが与えられて mが定まってしまえば，特定の R_1, R_2の値に対して Fを最小化するパラメータは kのみである．表 4.2は $m = 512$ とした時の Fの最小値とその時の最適化されたパラメータの値を示している．表 4.2 でプロトコル 3 と 4 の処理時間には事前計算に費やされる時間は含まれていないことに注意．表で各列は与えられた通信回線に対してサーバを 6 通りに変化させたとき得られる 6 通りの結果を示している．また異なる列は異なる通信速度に対応する．

表の各欄の記述法を説明するために，$(R_1, R_2) = (10^3, 10)$ に対応する欄の内容「P3/7/96」を取り上げる．最初の項目 P3 はこの時 4 つのプロトコルの内，最も処理時間が短いのがプロトコル 3 であることを示す．2 番目の項目 7 は最適な窓サイズが 7 であることを示し，3 番目の項目 96 は正規化処理時間を示す．この場合，プロトコルの全処理時間はクライアントが 96 回剰余乗算を行うのに費やす時間と同じであることを意味する．

表 4.2から読み取れる一般的傾向をまとめると，通信速度が比較的遅いとき Yao 法に基づくプロトコル 1 や 3 が有利である．また通信速度が速い時，Knuth 法に基づくプロトコル 2 や 4 がより効率的である．またサーバの処理能力が大きい時プロトコル 3 はプロトコル 1 に比べて効率が良く，プロトコル 4 は 2 に比べて効率が良い．

表 4.2: 正規化処理時間

(最適なプロトコル/最適な窓サイズ k/処理時間 (F), 但し $m = 512$)

$R_1 \setminus R_2$	1	10	10^2	10^3	10^4	10^5
1	P1/5/776	P1/4/676	P2/5/650	P2/5/645	P2/5/645	P2/5/645
10	P3/5/297	P3/6/166	P4/7/145	P4/7/139	P4/5/139	P4/7/139
10^2	P3/6/244	P3/7/103	P4/9/76	P4/9/69	P4/9/68	P4/9/68
10^3	P3/6/239	P3/7/96	P4/10/65	P4/12/51	P4/12/49	P4/12/49
10^4	P3/6/238	P3/7/95	P4/10/63	P4/13/46	P4/14/41	P4/15/39
10^5	P3/6/238	P3/7/95	P4/10/63	P4/13/46	P4/16/37	P4/18/34

R_1: 正規化されたサーバの計算力, R_2: 正規化された回線速度

Note: プロトコル 3 と 4 の処理時間は事前計算に要する時間を含まない．

4.4.3 数値例

依頼計算は計算時間の短縮を目的としているから，まずクライアント単独でべき乗剰余計算をした場合にどれだけ時間が掛かるかを明かにしなければならない．法 Nが素数の場合と，法 Nが二つの素数 Pと Q の積である場合の二つの典型的な場合を考えよう．

まず，法が素数の場合，Knuth 法はクライアント単独でべき乗剰余計算するのに最も適したアルゴリズムの一つである．この時クライアントは $(2^k + \lceil m/k \rceil + m - 2)$ 回の剰余乗算をしなければならない．$k = 5, m = 512$ とすれば，その回数は 645 回となる．

法が合成数の場合，文献 [85] の方法の様に中国剰余定理を利用して法 P, 法 Q でそれぞれべき乗を求めるのが適当である．2 度のべき乗は Knuth 法に従って計算すると，総計算時間は次のように表せる．

$$G = 2R(m) + 2(2^k + \lceil m/2k \rceil - 2)M(m/2) + CRT(m) \tag{4.10}$$

ここで，$R(m)$ は m ビットの数を $m/2$ ビットの剰余に丸めるのに必要な時間を表し，$M(m/2)$ は $m/2$ ビット剰余乗算に必要な時間を表し，さらに $CRT(m)$ は二つの $m/2$ ビットの数を中国剰余定理に従って合成し，m ビットの数を得るのに掛かる時間を表す．G を評価するために，次の関係を仮定する．

$$M(m/2) \simeq (1/4)M(m) \tag{4.11}$$

$$R(m) \simeq (1/2)M(m/2) \simeq (1/8)M(m) \tag{4.12}$$

$$CRT(m) \simeq (3/2)M(m/2) \simeq (3/8)M(m) \tag{4.13}$$

すると，

$$G \simeq (2^{k-1} + \lceil m/2k \rceil/2 + \lceil m/2 \rceil/2 - 3/8)M(m) \tag{4.14}$$

となる．更に $m = 512, k = 5$ とすると，

$$G \simeq 169 \cdot M(512) \tag{4.15}$$

右辺はクライアントが 169 回 512 ビットの剰余乗算を行う時間で, べき乗を計算できるという意味を持つ.

以上の準備を考慮しながら, 次にプロトコルの具体的処理時間を見積もった. 表 4.3はその結果をまとめたもので, 次の 4 つの条件を仮定している.

(1) 法のサイズ : $m = 512$ ビット

(2) クライアントの処理能力 : $T_0 = 60/768$ 秒

(3) サーバの処理能力 : $R_1 = 10^3$

(4) 通信速度 : $R_2 = 10$

条件 (2) は中国剰余定理を利用せずに単純な二進法を用いてべき乗剰余計算を行ったとき, クライアントの処理時間は 60 秒かかるという仮定である (二進法を用いるとべき乗は平均で 768 回の剰余乗算で実現されるから). これは第 3 章の 3.4.3節で仮定した IC カードの処理能力の 2 ~ 3 倍の処理能力に相当する.

条件 (3) を条件 (2) と組合わせると, サーバは単純な 2 進法によるべき乗剰余計算を 60 ミリ秒で完了することがわかる. これを RSA 暗号のスループットに換算すると, $512/60 \cdot 10^{-3}$bps $= 8.5$kbps である. 第 2 章で試作した DSP ボードの処理能力は 4.5kbps だからそれと比べると約 2 倍の処理能力に相当する. 専用 LSI では現在でも達成可能な処理時間である.

条件 (4) を条件 (2) と合わせて, 実際の通信速度に換算すると $m \cdot R_2/T_0 = 66$ (kbps) となる. なお, ISO 規格の IC カードの通信速度は 9.6kbps であり, それに比べるとかなり高い通信速度であるが, ここでは将来の通信機能の性能向上を見越してこの値を例にとる.

表 4.3: 数値例 ($m = 512$)

Method	Time (sec)	Comments
Client alone I	$768 \cdot T_0 = 60.0$	Binary
Client alone II	$645 \cdot T_0 = 50.4$	Knuth
Client alone III	$169 \cdot T_0 = 13.2$	Knuth and CRT
Quisquater, et al.	$271 \cdot T_0 = 21.2$	See Appendix
Proposed	$96 \cdot T_0 = 7.5$	Protocol 3
Matsumoto, et al.	1.5	10^{-30} security, CRT

$(R_1, R_2)=(10^3, 10)$, $T_0 = 60/768$(sec)

表 4.3は提案方式が同じ安全性条件の他の方式 (Client alone I ~III および Quisquater らの方式) と比べ最も処理時間が短いことを示している. Quisquater らによって提案された方式の性能解析は付録 C.1参照. 松本らのプロトコルの処理時間は 3章の解析結果に基づいて見積もった. 文献 [81] ではこの処理時間は加算連鎖の考え方を利用することによって更に短縮されうることが指摘されている. 松本らのプロトコルは 10^{30}回の総当たりによって秘密指数 Dが暴かれること, これは同じ表の他の方式に比べて非常に弱い安全性であると見ることができることに注意が必要である. 最近文献 [84] でこの総当たりの回数を下げる攻撃法が提案された.

4.5 考察

この節では計算結果の検算, 提案したプロトコルにわずかに修正を加えたプロトコルへの攻撃, プロトコルへの中国剰余定理の適用の 3 点について考察する.

提案プロトコルではサーバが正しい動作をしなくても秘密 Dは漏れないが, 計算結果の正しさは保証されない. RSA 暗号の署名作成にプロトコルを用いる場合であれば, 署名検査指数 Eを充分小さく設定しておけば結果の検算は容易である. クライアントが計算結果を自分自身で E乗して元のメッセージ Mと一致することが確認されれば, 結果は正しい.

提案プロトコルはクライアントが計算結果 Sをクライアント以外に開示しない限りは秘密指数 Dに関し零知識であり, 安全である. 実際にプロトコルを利用するシステムではこの条件に近づけるために幾つかの現実的な対策をとらなければならないが, そうした対策をなおかいくぐり, Sを入手して攻撃に利用しようとする攻撃者があるかもしれない. それ故, 計算結果 S をプロトコル終了後クライアントがサーバに渡すような若干の変更を加えた「修正プロトコル」の安全性を考察して置くのは意味のある事である. この修正プロトコルではべき指数 Dに関する何らかの知識が漏れていると思われる. 例えば, 不正なサーバが何らかの正しくない値をクライアントに返し, その影響を受けた S'を入手することによって指数 Dを導く攻撃が有り得る. このような攻撃 (能動攻撃) の一つが文献 [5] で検討された. ただし, 能動攻撃が S'の値そのものの入手を前提としている場合には, 検算を導入し正しい計算結果だけをサーバに返すようにすれば防止できる. しかし, より巧妙な能動攻撃として, サーバのプロトコルからの逸脱が計算結果に現れ検算で検出されたか否かの情報を用いて Dを導くような攻撃法もありうる. この種の攻撃法は松本らの依頼計算に対して最初に発見された [84]. 提案プロトコルに対して具体例を示すことは次の章の課題として残しておくが, 次の例はこの予想を支持している. プロトコル 1 においてサーバは y_iの値を一つだけ正しくない値に置き換えて返す. 例えば y_i'が正しくない値だとする. もしこれがクライアントによって検出されたならば, サーバは $d_i \neq 0$ であると推定できる. また検出されなければ $d_i = 0$ と推定できる. このような攻撃法に対して「修正プロトコル」は有効な対策を持たない. これまでのところ, サーバに計算結果を渡しても能動攻撃を防止でき, しかも効率の良いプロトコルが構成できるか否かは未解決問題である.

最後に文献 [75] で指摘されている様に, 法 Nが合成数の場合には中国剰余定理をクライアントの処理に適用して一層の処理時間の短縮をはかることが可能である. 仮に法 Nが素数 P, Q の積であるとすると, べき指数 Dを二つの指数, $D_P = D \bmod (P-1)$, $D_Q = D \bmod (Q-1)$ に分けて処理することができる. クライアントは D_P乗、D_Q乗を計算するためにプロトコルを利用するが, 同じ y_iを両方のべき乗計算に利用できる. また 2 つのべき指数は元のべき指数のほぼ半分のビット数なのでサーバは元のプロトコルの半数の y_iを計算するだけでよい. その結果サーバの乗算と通信量の両方が減らせるのである. 詳細な検討は文献 [41] に示されている.

4.6 むすび

この章では, べき乗計算のための 4 つの依頼計算プロトコルを提案し性能解析をした. 提案したプロトコルはクライアントが用いているべき指数 D を決して漏らすことが無い. さらに典型的なパラメータに対して, 提案プロトコルは従来提案されている同じ安全性を実現するプロトコルの内, 最も短い時間でべき乗を計算できる. 提案プロトコルは Yao 法, Knuth 法という効率の良いべき乗計算アルゴリズムに基づいて構成されているので, 提案プロトコルを大きく上回る性能を持つプロトコルを構成するには, 全く異なる方針でプロトコルを設計しない限り困難であると予想される. クライアントが計算結果をサーバに漏らすような依頼計算における安全性の検討と, そのような条件の下でも安全なプロトコルを構成することがここまでの議論では未解決の課題である.

第 5 章

正誤情報の通報による秘密の漏洩解析

5.1 はじめに

べき乗剰余計算の高速化を目的とした依頼計算法を検討し, 前章では秘密指数に関して新たな知識は一切もらさない依頼計算法を提案したが, その零知識性が証明できたのはクライアントが計算結果が正しかったか否かをサーバに知らせないという条件が満足されている場合だけであった[1]. さらに, 前章ではもしこの条件が満たされずクライアントがサーバに計算結果が正しいか否かを知らせると, 能動的な攻撃によって秘密指数に関する何らかの情報が漏れる可能性が指摘された. ただし, 秘密指数を現実的に導き出してしまう攻撃法が構成できるかどうかは前章までは未解決だった. この章ではクライアントが計算結果に関する情報を一切サーバに知らせないという理想化された条件が満たされない場合について, 安全性を考察する. この章で論じるのは攻撃的側面であるため依頼計算に対し否定的印象を与えるかも知れないが, 我々の目的は能動攻撃を詳細に検討する事によってプロトコルの性能や限界を明らかにし, 安心して使える依頼計算プロトコルを構築する事にある.

まず 5.2節では検討する依頼計算プロトコル（前節のプロトコル 3 ）の手順を示す. 5.3節でははじめにクライアントが正誤情報をサーバに送った場合, それが運び得る情報量を考察し, 次にその情報を引き出すためにサーバが採り得るプロトコルからの逸脱を幾つか示し, さらに具体的な能動攻撃の手順を示す. 次の 5.4節では考察を行い, 攻撃に対する現実的な対策例を示す. 最後に 5.5節で本章を結ぶ.

5.2 べき乗剰余のための依頼計算プロトコル

まず, 前節で提案した RSA 用依頼計算のひとつであるプロトコル 3 の流れを示す. 前節の記述と同じであるが, 計算結果の扱いの部分が後の考察に備えて若干細かくなっている.

[1] 計算結果そのものを渡さず, 計算結果の正誤を知らせる 1 ビットのメッセージをサーバに返すとうい極めて微妙なプロトコルの変更を加えただけで零知識性は保証されない, という意味であることに注意. ただし, そのような場合でも攻撃が受動攻撃ならば計算結果は常に正しいので系列のシミュレートが可能で安全性は保証される.

(E,N) を RSA 暗号の公開情報, (D,P,Q) を秘密情報とする. Nは素数 P,Q の積であり, Eと D は $E\cdot D=1 \bmod L$ を満たす. ただし, L は $(P-1)$, $(Q-1)$ の最小公倍数である. Dは基底 b を用いて $D=\sum_{i=0}^{l-1} d_i\cdot b^i$と表されているとする. プロトコル 3 の目的は $S=M^D \bmod N$を計算することである. プロトコルの説明で「クライアント $\rightarrow$ サーバ: M, N」といった記述を用いるがこれはクライアントがサーバに M,Nを送るということを意味する. Z_Nおよび Z_N^*で, それぞれ Nを法とする整数環と, Z_Nの要素の内 Nと互いに素なものからなる乗法群を表す.

[プロトコル 3]

準備： クライアントは $(b-1)$ 個の乱数 $R_j\in Z_N^*$を生成し,

$$R^{-1}=\frac{1}{\prod_{j=1}^{b-1} R_j^j} \bmod N \tag{5.1}$$

を計算する.

ステップ 1： クライアント $\rightarrow$ サーバ： M, N

ステップ 2： サーバ $\rightarrow$ クライアント： $\boldsymbol{X}=[X_1,X_2,\cdots,X_i,\cdots,X_{l-1}]$. 但し, $X_i=M^{b^i} \bmod N$

ステップ 3： クライアント $\rightarrow$ サーバ： $\boldsymbol{Y}=[Y_1,Y_2,\cdots,Y_j,\cdots,Y_{b-1}]$。但し,

$$\begin{aligned} Y_j &= R_j\cdot \prod_{i \text{ s.t. } d_i=j} X_i \bmod N \\ X_0 &= M \end{aligned}$$

ステップ 4： サーバ $\rightarrow$ クライアント： $Z=\prod_{j=1}^{b-1} Y_j^j \bmod N$

ステップ 5： クライアントは $S=Z\cdot R^{-1} \bmod N$を計算する. もし $M=S^E \bmod N$ならば, 結果 (OK,S) を結果 $\mathcal{O}$とする. それ以外（結果が正しくない時）は NG を結果 $\mathcal{O}$とする（この時, 誤った署名$\tilde{S}$は返さない）. （終）

公開のべき指数 Eは充分小さく選んでおくものとし, ステップ 5 の検算に要する時間は無視できるものとする. 準備の計算はメッセージと独立なのでプロトコル開始前に済ますことができる.

プロトコル 3 は次に挙げる（1）, （2）いずれの状況でも D に関する新たな知識を漏らさないという, 顕著な特性を持つ.

（1） 能動攻撃が行われても, クライアントが結果 $\mathcal{O}$を他者に開示しない場合.

（2） クライアントが結果 $\mathcal{O}$を他者に開示しても, 受動攻撃しか行われない事がわかっている場合.

ここで受動攻撃とはサーバがプロトコルの仕様から逸脱することが無い攻撃を指す. いずれにしても, クライアントが計算結果の正誤を漏らさない限りはクライアントの秘密は保たれる. ではクライアントが正誤を漏らした場合, それが Dに関しどの程度の情報量を担っているのであろうか? 次の節では, この問題を検討する.

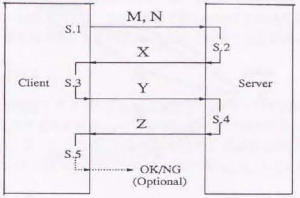


図 5.1: 基本プロトコル

5.3 情報の漏洩

5.3.1 Dに関する問い合わせとしてのサーバの逸脱

当面, サーバの不正（プロトコル仕様からの逸脱）はステップ 2 でのみ行われるものとし, サーバが計算したベクトル$\boldsymbol{X}$には少なくとも一つは正しくない値が含まれているものとする. サーバはステップ 4 でも不正を働き得るが, その点は後の節で考察する. 攻撃者の観点に立てば, 正しくないベクトル$\boldsymbol{X}$は秘密指数 Dに関するクライアントへの問い合わせと見ることができる.

i 回目の問い合わせベクトル$\boldsymbol{X}_i$が具体的に与えられたとき, Dの値を一つ定めると対応するクライアントの応答は OKか NG のいずれかに確定する. そこで Dの値を, $\boldsymbol{X}_i$と NG を結ぶ弧一本に対応させるという規則で, クライアントが使用している可能性のある全ての Dの値と弧を一対一対応に描けたとする. 例えば特定の値 D_1を使ってプロトコルを実行した時に, NGが返されるならば, この D_1の値に対応して$\boldsymbol{X}_i$と NG を結ぶ弧をひとつ描く. Dの値を次々に変えてこの操作を続けて行くと, 図 5.2のようなグラフが得られる. Dの候補の数が大きい時, 全ての Dについてこの様な弧を書くことは現実的には不可能だが, あくまで思考実験的には常にこの様な図が一意に定まる. 図 5.2では $(i-1)$ 回目の問い合わせを終わった段階での可能な Dの候補の数が N_i, この内クライアントが OK と答えるものが pN_i個、NG と答えるものが $(1-p)N_i$個あるとしている. 一度も問い合わせをしていない時, Z_L^*の任意の要素が Dである可能性があるから, 初期値は $N_1=|Z_L^*|$である. ただし, $|S|$ は集合 Sの要素数を表す. さて, 次は実際にプロトコルを走らせて$\boldsymbol{X}_i$に対応した応答として, 例えば OKが観測されたとする. この時, NG に対応する Dの候補は全て, クライアントが使用していないものとして候補から除外することができる. 残った候補について, 問い合わせと応答の観測を続けて行けば Dの候補は限定されて行き, 何回かの観測の後には真の Dを特定できるかも知れない.

次にこの様な攻撃法を情報理論的に解析するために, いくつかの概念を定義する. まず OK確率,

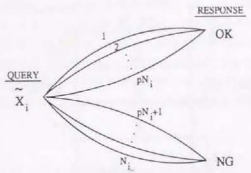


図 5.2: サーバへの問い合わせとその応答

NG 確率をそれぞれ次のように定義する.

$$Prob(\mathcal{O}_i = OK) = (OK\text{を出力する } D\text{の数})/N_i = p \tag{5.2}$$

$$Prob(\mathcal{O}_i = NG) = (NG\text{を出力する } D\text{の数})/N_i = 1-p \tag{5.3}$$

ここで $\mathcal{O}_i$ はクライアントからの i 番目の応答を表す. なお p は定数ではなく, それまでに使われた問い合わせの履歴とクライアントが使っている Dの値とに依存することに注意. ただし, ここでは表現を簡潔にするためにその事を式の中であらわには示していない. なお付録 D.1ではそれがあらわにわかる表現, 例えば $P(0|O_{i-1})$ を使用している.

また, Dの equivocation を文献 [55] の定義にならい, 次のように定める.

$$H(D|O_i) = E[\log_2 N_{i+1}] \tag{5.4}$$

ここで O_i は i 回目までのクライアントの応答 $\mathcal{O}$の系列を表し, 期待値は与えられた問い合わせの系列と, 全ての可能な Dの値に対してとる. equivocation と等価な評価量として, 次式で定義される相互情報量 $I(D;\mathcal{O}_i)$ を用いても良い.

$$\begin{aligned} I(D;\mathcal{O}_i) &= H(D) - H(D|\mathcal{O}_i) \\ &= \log_2 N_0 - E[\log_2 N_{i+1}] \end{aligned} \tag{5.5}$$

$I(D;\mathcal{O}_i)$ は $\mathcal{O}$を i 回観測したときに Dに関してどれだけの情報が得られるかを表す量である. 与えられた問い合わせの系列に対して, 次の不等式を証明できる (付録 D.1参照).

$$I(D;\mathcal{O}_i) - I(D;\mathcal{O}_{i-1}) = E[h(p)] \leq 1 \tag{5.6}$$

ここで, $h(x)$ はエントロピー関数であり, $h(\boldsymbol{x}) = -x\log_2 x - (1-x)\log_2(1-x)$. この不等式は 1 回の応答の観測で得られる Dに関する情報量は, 高々 1 ビットであることを意味している.

もし, 問い合わせを行う度に Dの候補の数を丁度半数に限定できるような, 問い合わせの系列 $\overline{\boldsymbol{X}}_i$ が存在するならば, 最も少ない問い合わせ回数で正しい Dを特定することができ, $\lceil \log_2 N_0 \rceil \simeq 512$ 回

の問い合わせですむ. 以上の議論は依頼計算プロトコルの個別の仕様に依存しておらず, 安全性の証明付き依頼計算に対して一般に成り立つ. なお, 有効な問い合わせの形は具体的なプロトコルの仕様によって変わるので, 有効な問い合わせ系列 $\overline{\boldsymbol{X}}_i$ は発見的に探索する必要がある.

5.3.2 問い合わせベクトル

前節では Dに関する情報をクライアントから引き出すため, サーバが問い合わせベクトルをクライアントに送るという見方で攻撃を考察し, 一度の問い合わせで得られる Dに関する情報量は高々 1 ビットであることを示した. ここではプロトコル 3 において有効な問い合わせベクトルとして, $\boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3$を導入する. これらはそれぞれ, Dの第 s,t 桁の値 d_s, d_t $(1 \leq s,t \leq l-1)$ に対して,

- d_sが偶数か否か,
- d_sが零か否か,
- d_sと d_tが等しいか否か

をクライアントに問い合わせるためのベクトルである. 各ベクトルの仕様を表 5.1にまとめた. 表の第 1 列のベクトル表現において, 例えば $-X_s$はプロトコルに従えば X_sを返すべき所を $N-X_s$を返す事を意味しており, その他の省略されている要素は, プロトコルに従って正しく計算されていることを表す. $b=16$ で Dが 512 ビットとすると $l=128$ となり, $1 \leq s,t \leq 127$ である.

表 5.1: 問合せベクトル

Vector	Decision Rule		
	OK	NG	OK Prob.
$\boldsymbol{V}_1(s) = [\cdots, -X_s, \cdots]$	d_s =even	d_s =odd	1/2
$\boldsymbol{V}_2(s) = [\cdots, X'_s, \cdots]$	$d_s = 0$	$d_s \neq 0$	1/16
$\boldsymbol{V}_3(s,t) = [\cdots, X_s T_\cdots, X_t T^{-1} \cdots]$	$d_s = d_t$	$d_s \neq d_t$	1/16

Note 1: $-X_s$ means $(N-X_s)$. X'_s means a value not equal to X_s.
Note 2: $T \in Z_N$ and $T = M$ is recommended.
Note 3: OK probabilities are for $b = 16$.

表の第 2, 3 列は問い合わせ結果 (OKまたは NG) を得たときの判断法を示しているが, この判断の根拠は, 正規のプロトコルを行った時, Sが X_iによって次のように表現できることにある.

$$S = \prod_{i=0}^{i-1} X_i^{d_i} \bmod N \tag{5.7}$$

この式の右辺で, $\boldsymbol{X}$を $\boldsymbol{V}_1(s)$ に置き換えて得られる値を $\tilde{S}$であらわすと, Sと $\tilde{S}$の関係は,

$$\tilde{S} = (-1)^{d_i} \cdot S \bmod N \tag{5.8}$$

従って次のような判断が可能となる.

- $\mathcal{O} = OK$ ならば　d_sは偶数
- $\mathcal{O} = NG$ ならば　d_sは奇数

他のベクトルの判断基準も同様にして得られる.

なお, V_2に関しては, 判断を誤らせる X'_iの値がわずかながら存在する. たとえば, $X'_i = -X_s$はV_2の条件を満たしているが, 実はV_1でもあるので d_sが偶数の時, V_2として判断すると誤りとなる. しかしこのような例は希である. $V_3(s,t)$ に対しても例えば $T = 1$ というような希な例外が存在する.

以上に説明してきた3つのベクトル以外にも例えば d_sと d_tの比を決める様な問い合わせベクトルを構成できる. さらに, これまでは1つないし2つの要素に変更を加えたベクトルのみを考えてきたが, 3要素以上に変更を加えたベクトルも構成できよう. しかし, 問も無く示すように, 上記3つのベクトルだけを用いて能率のよい攻撃法を構成できるので, ここではこれ以上新しいベクトルは導入しない.

5.3.3 Dの導出

ここでは前節で導入した3つの問い合わせベクトルを用いた攻撃手順の例を示す. $b = 16$ とすると, $0 \le d_i \le 15$. 従って攻撃手順の目的はまず, d_i　$(i = 1, 2, \cdots, 127)$ をその値に従って16通りの部分集合に振り分けることである. プロトコルの仕様によって, 攻撃者は d_0に関する問い合わせをクライアントにすることができないことに注意. しかし, d_0は奇数であることはわかっているので, 8通りの可能な値 $(1, 3, 5, \cdots, 15)$ のどれかであり, 総当りにより決定できる.

[攻撃手順]

ステップ1： $V_1(s)$ を $s = 1, 2, \cdots, 127$ についてそれぞれクライアントに送る. これによって各 d_sの偶奇が決まり, 添え字 s は次のような二つの集合に分かれる.

$$S_e = \{s | s \ s.t. \ d_s \text{が偶数}\}$$
$$S_o = \{s | s \ s.t. \ d_s \text{が奇数}\}$$

ステップ2： S_eに属する s, t $(s \neq t)$ について, $V_3(s,t)$ をクライアントに送る. その結果 S_eは d_sの値に応じて最大8つの部分集合に分けられる. 同様に S_oも最大8つの部分集合に分けられる.

ステップ3： S_eの8つの部分集合の各々からとりだした8つの代表要素 s について, $V_2(s)$ を作ってクライアントに送り, どれが $d_s = 0$ に対応する集合かを決める.

ステップ4： S_eの残る7つの部分集合と, S_oの8つの部分集合が, それぞれどの d_iの値に対応するかが不明であり, $7! \cdot 8!$通りの割当方が存在する. さらに d_0の値は8通り可能であるから, この段階で正しい Dの候補は $8 \cdot 7! \cdot 8! = (8!)^2$通り存在する. 可能な候補を $\tilde{D}$と書く時どの$\tilde{D}$ が正しいかを総当たりで検証する. 正しいか否かは公開鍵 Eを用いて法 Nの下で適当な底 a を$\tilde{D}$乗し E乗して元も戻るかどうかを調べればクライアントを利用する事無く確かめられる.　(攻撃手順終)

各ステップで必要となる計算量を表5.2にまとめた. クライアントへの問い合わせ回数は680回, ステップ4の総当たりの計算の回数は約 1.5×2^{30}であり実現不可能な回数ではない. ステップ3までで考えると, $I(D; \mathcal{O}) = 512 - \log_2(8!)^2 \simeq 481$ であり, 攻撃者は Dに関して481ビットの情報を引きだしたと考えられる.

図5.3は問い合わせ回数の増加に対するequivocationの減少を, その下限 $H(D) - i$ と共に示している. 下限との比較で提案した攻撃法が $b = 16$ に対しては能率的であることがわかる.

表 5.2: 攻撃に必要な計算量

Step	# of Queries	Comments
1	127	By V_1
2	546	By V_3, See Appendix D.2
3	7	By V_2
4	0	1.5×2^{36} off line Processing

Note: $1.5 \times 2^{30} \simeq (8!)^2$

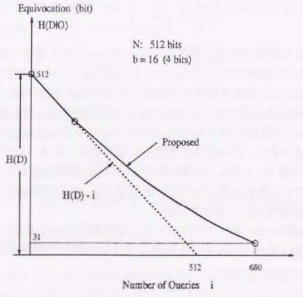


図 5.3: 秘密鍵 Dの equivocation の減少

5.3.4 $H(O|D)$ 対 b のトレードオフ

攻撃手順のステップ 3 を終えた段階での equivocation の値は, b のサイズに依存している（ここで、b は Dを表すのに用いた基底である）. 例えば b を最小の 2 に選んだ場合, ステップ 1 を終えた段階での equivocation は常であり, 512 回の問い合わせを行うだけで Dは一意に決定できる. すなわちこの場合には図 5.3における下限が達成される訳である. また, b のサイズを線形に増大させると, ステップ 3 を終えた段階で equivocation は急激に増大すると考えられる. この傾向を詳細に見ることにしよう.

$b \log_2 b \ll m$ の場合（但し, $m = \log_2 N$）, equivocation は次式で近似される.

$$2 \log_2 \left(\frac{b}{2}! \right) \tag{5.9}$$

その理由は, この場合には S_eと S_oの各部分集合が空集合である可能性が低く, 従って攻撃のステップ 4 での総当たりの回数が $(b/2)!^2$と見積もれるからである.

また, $b \log_2 b \gg m$ の場合, 次式で近似される.

$$2 \log_2 \left(\tfrac{b}{2} P_{\frac{m}{2 \log_2 b}} \right) \tag{5.10}$$

ここで${}_nP_k = n!/(n-k)!$である. この様に近似される理由は, $b \log_2 b \gg m$ の場合には S_e, S_oの各部分集合は, 空集合ないしただ一つの要素を持つ傾向があるからである. 式 (5.9),(5.10) において, 括弧内の値はいずれも $|S_e| = m/2 \log_2 b$ 個の要素を $b/2$ 個の入れ物に分配する場合の数を表している. 充分に大きい b のサイズに対しては式 (5.10) はさらに, 次式により近似される.

$$m \left(1 - \frac{1}{\log_2 b} \right) \tag{5.11}$$

この式はステップ 1 を終えた段階での Dに関する equivocation を表す式と一致している. もともと式 (5.10) は, ステップ 3 を終えた段階での equivocation を定式化したものだから, bが大きい時に (5.10) が (5.11) で近似されるということは, b のサイズの増大とともにステップ 2, 3 でサーバが引き出すことのできる情報量は無視できる程小さくなるということを意味する. 以上の結果を図 5.4に示す.

図によれば前節で示した攻撃法に関する限り, 充分大きい b を選ぶことによって, 安全性を確保することが可能となると考えられる. その意味で提案した攻撃法は b のサイズが比較的小さいという限定された場面でのみ有効である. ただし, 依頼計算プロトコルの処理速度は b のサイズに依存するので, 処理速度を度外視して b を大きく選べないのも事実である. なお処理速度を最小にするという意味で最適な b のサイズは前章で示した通りである.

以上の議論は 3 種類の問い合わせベクトルのみ用いる攻撃法のみを念頭に置いたものである. 一般の攻撃法に対しても安全な b のサイズが存在するかどうかは未解決の課題である.

5.4 考察

ここでは依頼計算プロトコルのステップ 4 における不正と, 攻撃に対する現実的な対策について考察する.

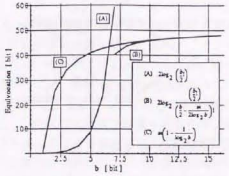


図 5.4: Equivocation 対 b のビット長

5.4.1 ステップ 4 での不正について

ステップ 4 で攻撃者が不正をしなかった場合のクライアントの応答を $\mathcal{O}$で表し, ステップ 4 で不正が有った場合のクライアントの応答を $\mathcal{O}'$で表す. 図 5.5は相互の関係を表す図である. ステップ 4 で不正をしなかったならば OKが返されるはずのところが, ステップ 4 での不正によって $\mathcal{O}'$では OK が返される場合と NGが返される場合に分かれる. $\mathcal{O}$が NG の場合も同じである.

まず, 攻撃者は M, N, $\boldsymbol{X}$, $\boldsymbol{Y}$ から Dに関する情報を引き出す事は現実的に不可能である. これは前章で多項式時間識別不可能性を用いて示した. 従って, 図 5.5において $\mathcal{O} = OK$が $\mathcal{O}' = OK$ に置き変わる確率は Dの値と独立である. すなわち,

$$Prob(\mathcal{O}'|\mathcal{O}, D) = Prob(\mathcal{O}'|\mathcal{O}) \tag{5.12}$$

これは, Dから $\mathcal{O}$を経て $\mathcal{O}'$へ達する情報の流れがカスケードであることを示している. カスケードシステムでは情報量は途中で減る事はあっても増える事はないことが知られている（例えば文献 [59] のページ 302 参照）. すなわち,

$$I(D;\mathcal{O}) \geq I(D;\mathcal{O}') \tag{5.13}$$

これはステップ 4 での不正は Dに関する情報量を増やすことには役立たない事を示している.

5.4.2 現実的な対策の例

ここでは能動攻撃をされても Dに関する漏洩情報量をある程度に制限する対策の例を示す. ここで紹介するのは, NG の判定の許容回数をある値 (u) に制限する方法であり, もし, NG を u 回観測したら同じ Dに対してはプロトコルを行わない様にするという方法である. この対策を施した場合の平均漏洩情報量を求めよう. その際、平均は可能なすべての Dの値が一様な確率で選ばれるものとして求める. まず, すべての問い合わせベクトルは等しい OK確率 p を持つものと仮定する. 従っ

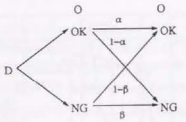


図 5.5: イベントツリー

て NG 確率も一定で $q = 1 - p$ であるとする. 今, 攻撃者はクライアントに対し, 充分多数回である t 回にわたって問い合わせを行ったとする. すると,

$$I(D;O) = \sum_{i=0}^{t} i \cdot P_i + t \cdot \left(1 - \sum_{i=0}^{t} P_i\right) \tag{5.14}$$

$$\simeq \sum_{i=0}^{\infty} i \cdot P_i \tag{5.15}$$

$$= \frac{u}{q} \tag{5.16}$$

ここで, P_iは負の指数分布を表し, ここでは次のように定義する.

$$P_i = \begin{cases} 0 & (\text{for } i < u) \\ \binom{i-1}{i-u} q^u p^{i-u} & (\text{for } i \geq u) \end{cases} \tag{5.17}$$

式 (5.14) の導出は付録 D.3を参照. ここまでは OK確率が一定であるとしたが, u を適当に小さく選ぶ限りその仮定が妥当であることを次に示そう. 式 (5.15) では u/qよりも充分大きいパラメータ lを持つ項は他の主要な項に比べ無視できるから, $I(D;O)$ の値を決める主要な項は $i \approx u/q$までのパラメータを持つ項である. 一方, 最大の OK確率 0.5 を持つ問い合わせベクトル $\boldsymbol{V}_1$は $512/k$通りある. また攻撃者が $I(D;O)$ を最大にするには OK確率が大きいベクトルから順に問い合わせに使えば良い. 従って $512/k \gg u/q$であるように uが選ばれていれば, OK確率は一定 (0.5) であると考えて良い. このことから pが一定であるという仮定の下で導かれた式 (5.16) を $I(D;O)$ の評価に用いる事ができて,

$$I(D;O) \simeq 2u \tag{5.18}$$

が成り立つ. これは平均漏洩情報量を u の 2 倍に抑えることができること示している. 仮に $u = 5$ とすると平均情報漏洩量は 10 ビットであり, 標準偏差は 2.2 ビットである (付録 D.3参照).

5.5 むすび

この章では, 前節で提案された RSA 用依頼計算プロトコルは計算結果をサーバに渡さないという仮定が重要な意味をを持ち, もしそれが守られない場合にはクライアントの秘密が漏れる可能性がある事を示した. ここでの教訓としては, 安全なプロトコルでもそれに微妙な変更を加えるだけで安全で無くなってしまうことがあるということ, また, 安全性の証明のないプロトコルを, 充分な安全性の検証の無いまま用いることは大きなリスクを伴うことであることが挙げられる.

このような攻撃を防ぐために, この章では簡単な対策を示したが, 本質的に安全なプロトコルを求めて, 次の章ではさらなるプロトコルの改良を検討する. 能動攻撃に対する依頼計算プロトコルの安全性を高めるために, 松本らは文献 [86] で同じプロトコルを 2 回用いて計算をするという案を提案し, S2プロトコル [75] に適用した. 我々もその案を利用し, 提案プロトコルの安全性を高める事を検討している. 我々の場合, 安全性についてはできるだけ厳密な証明付きで保証しようとしている点で文献 [86] のアプローチとは立場が異なっている.

第 6 章

正誤情報の通報にも強い依頼計算

6.1 はじめに

前章ではべき乗の為の依頼計算プロトコルにおいて, べき乗結果 (より正確にはべき乗結果が正しかったか否かの検算結果) をサーバに渡すことは, プロトコルの安全性に重大な影響をおよぼすことを見てきた. この章では, べき乗結果をサーバに渡してもある程度安全性を保てるようにプロトコルを改良し, その効果をできる限り厳密に評価する. 依頼計算研究の最終的な目標はあくまで結果をサーバに渡しても証明付きで安全な依頼計算を構成すること, またその安全性を保ちながらどれだけ計算速度の速い効率のよいプロトコルを構成できるかを示すことにあるが, 様々な観点からの考察にも関わらずこの目標にはまだ達していない. 本章では可能な攻撃パターンを 3 種類に分類し, その内の 2 つに対しては証明付きで安全性が保証できることを示す.

6.2 改良版プロトコル

RSA 暗号の署名計算に適用される依頼計算において, 能動攻撃に対する安全性を高めるために, 松本らは対抗手段として秘密のべき指数を 2 つの整数の積に分解して, 基本となるべき乗プロトコルを 2 回シリアルに走らせるという方法を提案した [86]. 我々も前節で検討したプロトコル 3 にこのアイディアを適用し, その安全性を検討することにする. 以下, 改良版プロトコルをプロトコル 5 と呼ぶ. ここでは法 N の素因数 P, Q を $P = 2P' + 1, Q = 2Q' + 1$ (P', Q' は素数) の形に限定する.

[プロトコル 5]

準備： クライアントは乱数 A を Z_L^* から選び, $D = A \cdot B \bmod L$ を満たす様に B を決める. さらに, $2(b-1)$ 個の乱数 $R_{1,j}$, $R_{2,j}$ を Z_N^* から選び

$$R^{-1} = \frac{1}{\left(\prod_{j=1}^{b-1} R_{1,j}^j\right)^B \left(\prod_{j=1}^{b-1} R_{2,j}^j\right)} \bmod N$$

を計算する.

ステップ 1： クライアント → サーバ： M, N

ステップ 2： サーバ → クライアント： $\boldsymbol{X}_1 = [\cdots, X_{1,i}, \cdots]$. ここで,

$$X_{1,i} = M^{b^i} \bmod N \quad (i = 1, \cdots, l-1)$$

ステップ 3： クライアント → サーバ： $\boldsymbol{Y}_1 = [\cdots, Y_{1,j}, \cdots]$. ここで,

$$\begin{aligned} Y_{1,j} &= R_{1,j} \cdot \prod_{i \text{ s.t. } a_i = j} X_{1,i} \bmod N \quad (j = 1, \cdots, b-1) \\ X_{1,0} &= M \end{aligned}$$

ステップ 4： サーバは $W = \prod_{j=1}^{b-1} Y_{1,j}^j \bmod N$ を計算する.

ステップ 5： サーバ → クライアント： $\boldsymbol{X}_2 = [\cdots, X_{2,i}, \cdots]$. ここで,

$$X_{2,i} = W^{b^i} \bmod N \quad (i = 0, \cdots, l-1)$$

ステップ 6： クライアント → サーバ： $\boldsymbol{Y}_2 = [\cdots, Y_{2,j}, \cdots]$. ここで,

$$Y_{2,j} = R_{2,j} \cdot \prod_{i \text{ s.t. } b_i = j} X_{2,i} \bmod N \quad (j = 1, \cdots, b-1)$$

ステップ 7： サーバ → クライアント： $Z = \prod_{j=1}^{b-1} Y_{2,j}^j \bmod N$

ステップ 8： クライアントは $S = Z \cdot R^{-1} \bmod N$ を計算し, 結果 $\mathcal{O}$ をサーバに送る. 但し, 結果は次の手順で求める.

もし, $M = S^E \bmod N$ が成り立てば, $\mathcal{O} = (OK, S)$.

また, $M = N - S^E \bmod N$ ならば, $\mathcal{O} = (OK, N - S)$.

それ以外は, $\mathcal{O} = NG$ **(終)**

改良版ではクライアントは $D = A \cdot B \bmod N$ を満たす二つの秘密の変数 A, B を用いている. ステップ 2 から 4 は M の A 乗を計算する手続き, ステップ 5 から 7 は W の B 乗を計算する手続きである. プロトコル 3 の場合と同様に, 準備の段階で必要となる計算時間は除外して考えて良く, また E は充分小さいとする. ステップ 8 の検算は符号の正負を入れ換えるだけのサーバの不正を排除するように設計されている.

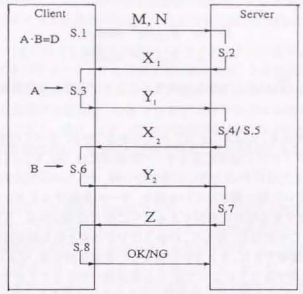


図 6.1: 改良版依頼計算プロトコル

6.3 安全性

6.3.1 攻撃の定義と分類

プロトコル 3 に対する攻撃同様に, D を推定するのに役立つ情報を担っているのは $\mathcal{O}$ だけであり, $\mathcal{O}$ が攻撃者に渡らなければプロトコルを通して漏れる知識は一切ないし, また受動攻撃に対しては $\mathcal{O}$ が攻撃者に渡っても安全である. しかし, 能動攻撃によって $\mathcal{O}$ から D に関する知識が漏れる可能性がある.

サーバが不正を行えるのは, ステップ 2, 4／5, 7 の 3 カ所である (図 6.1参照). この内、ステップ 7 での不正は前章で説明したように, A および B に関して漏れる情報を増やすのに役立たないから除外して良い. 従って不正の行われる場所としてはステップ 2 とステップ 4／5 のみを検討すれば良く, その内のいずれでサーバがプロトコルから逸脱するかによって攻撃を 3 クラスに分けることができる. クラスを定義するためにまず, 不正なサーバのプロトコルからの逸脱をベクトル $\boldsymbol{U}$ と $\boldsymbol{V}$ で定義する.

定義 2 与えられた b, M, N に対して, クライアントは常にプロトコルに従った計算を行ない, サーバは $\boldsymbol{X}_1$ と $\boldsymbol{X}_2$ の代わりにそれぞれ $\widetilde{\boldsymbol{X}}_1$ と $\widetilde{\boldsymbol{X}}_2$ を返したとする. この時, 攻撃者のプロトコルからの逸脱をベクトル

$$\begin{aligned} \boldsymbol{U} &= [U_1, U_2, \cdots, U_{l-1}] \\ \boldsymbol{V} &= [V_0, V_1, \cdots, V_{l-1}] \end{aligned}$$

で表し,

$$U_i = \widetilde{X}_{1,i}/M^{b'} \bmod N$$
$$V_i = \widetilde{X}_{2,i}/\widetilde{W}^{b'} \bmod N$$

と定義する. ここで$\widetilde{W}$は, $\widetilde{\boldsymbol{X}}_1$に応じて正しいクライアントが計算した結果を$\widetilde{\boldsymbol{Y}}_1$ と表すとき, $\widetilde{W} = \prod_{j=1}^{b-1} \widetilde{Y}_{1,j}^{\,j}$と定義する. □

U_iは$\widetilde{X}_{1,i}$と $X_{1,i}$の比であり, V_iは$\widetilde{X}_{2,i}$と $X_{2,i}$の比である. なお, すべての要素が 1 である l次元ベクトルを$\boldsymbol{I}_l$で表せば, プロトコルに忠実に従うサーバの場合には, $(\boldsymbol{U}, \boldsymbol{V}) = (\boldsymbol{I}_{i-1}, \boldsymbol{I}_i)$ となる.

ベクトル$\boldsymbol{U}$, $\boldsymbol{V}$は, Mまたは$\widetilde{W}$が Nと互いに素でない時, それらの逆元を計算できないので不定となる. しかし Mが Nと互いに素に選ばれている限り, サーバ単独で Nと互いに素でない要素を持つベクトル$\widetilde{\boldsymbol{X}}_1$や$\widetilde{\boldsymbol{X}}_2$ を生成できる確率は無視できるように法 Nは選ばれる. さもなければ Nの素因数分解が高い確率で行えることになる. 従って, $\boldsymbol{U}$および$\boldsymbol{V}$が不定となる確率は無視してよい. さらに, 仮にサーバが非常に低い確率であれ, Nと互いに素でない要素を持つ$\widetilde{\boldsymbol{X}}_1$や$\widetilde{\boldsymbol{X}}_2$ を作れた場合には, Nとその要素に互除法を適用することによって法 Nを素因数分解することができ, 公開鍵 Eから Dを求めることができるが, それはプロトコルが実行されたために洩れたわけではなく, プロトコル自体の安全性とは無関係である.

さて, 上に定義した逸脱ベクトルを用いてサーバの攻撃を 3 つのクラスに分ける.

クラス 1 : ステップ 2 のみで不正を行う場合 $(\boldsymbol{U} \neq \boldsymbol{I}_{l-1}, \boldsymbol{V} = \boldsymbol{I}_l)$.

クラス 2 : ステップ 4 ／ 5 でのみ不正を行う場合 $(\boldsymbol{U} = \boldsymbol{I}_{l-1}, \boldsymbol{V} \neq \boldsymbol{I}_l)$.

クラス 3 : ステップ 2 および 4 ／ 5 で不正を行う場合 $(\boldsymbol{U} \neq \boldsymbol{I}_{l-1}, \boldsymbol{V} \neq \boldsymbol{I}_l)$.

逸脱$\boldsymbol{U}$, $\boldsymbol{V}$ の結果, べき乗結果$\widetilde{S}$は, 次のようになる.

$$\widetilde{S} = S \cdot \left(\prod_{i=1}^{l-1} U_i^{a_i}\right)^B \cdot \prod_{j=0}^{l-1} V_j^{b_j} \bmod N \tag{6.1}$$
$$= S \cdot \alpha^B \beta \bmod N = S \cdot \epsilon \bmod N \tag{6.2}$$

ここで、

$$\alpha = \prod_{i=1}^{l-1} U_i^{a_i} \bmod N \tag{6.3}$$
$$\beta = \prod_{j=0}^{l-1} V_j^{b_j} \bmod N \tag{6.4}$$
$$\epsilon = \alpha^B \beta \bmod N \tag{6.5}$$

プロトコル 5 においてクライアントは, $\epsilon = \pm 1$ の時 OKを返し, それ以外の時には必ず NG を返す.

さて以上の準備の下クラス 1 , クラス 2 攻撃に対して, 次の定理を証明できる.

定理 2 任意の多項式時間のクラス 1 攻撃を行う攻撃者に対して, プロトコル 5 は秘密指数 Dに関して零知識である. 同様にクラス 2 攻撃に対してもプロトコル 5 は Dに関して零知識である. □

証明は, 任意のクラス 1 (または 2) 攻撃者と正しいクライアントが生成する通信系列のアンサンブル (法 Nのビットサイズをパラメータとする通信系列の確率分布) と多項式時間で区別できないアンサンブルを生成する確率的多項式時間シミュレータが構成できることを示せばよい. クラス 1 , 2 攻撃がシミュレートできる直感的な理由は, 変数 A および Bはそれぞれ単独で観測されたら完全な乱数と見なせるということである. 逆に, A と Bが対で観測されれば, もはや乱数と見ることはできない. このことはクラス 3 攻撃に対して安全性の証明が容易でない理由にもなっている.

[証明]
クラス 1 攻撃を行う不正なサーバ S'と正当なクライアント Cとの間の通信系列を, S'をサブルーチンとすることで多項式時間でシミュレートするシミュレータ M_1が構成できることを示す. S'はステップ 2 でのみプロトコルから逸脱するが, その逸脱ベクトルを$\boldsymbol{U}$で表す. ここで M, $X_{i,}$ $(i = 1, \cdots, l-1)$ は Nと互いに素であるとする. M, X_iが Nと互いに素の時, $\boldsymbol{Y}_1$, $\boldsymbol{X}_2$, $\boldsymbol{Y}_2$ も Nと互いに素であり, 特に, $\boldsymbol{Y}_1$, $\boldsymbol{Y}_2$ はそれ以前のステップでプロトコルが生成する系列の値によらず, それぞれ Z_N^*上の b 次元ベクトルから一様ランダムに生成される. なぜなら,

$$Y_{1,j} = R_{1,j} \cdot \prod_{i:a_i=j} X_{1,i} \bmod N (j = 1, \cdots, b-1\}$$
$$Y_{2,j} = R_{2,j} \cdot \prod_{i:b_i=j} X_{2,i} \bmod N (j = 1, \cdots, b-1)$$

において $R_{1,j}$, $R_{2,j}$は Z_N^*から一様ランダムに選ばれるからである. このことから秘密指数 Dに依存する可能性があるのは $\mathcal{O} \in \{OK, NG\}$ のみである. クラス 1 攻撃の結果得られる署名$\widetilde{S}$と正しい署名 Sの比を取ると,

$$\epsilon = \widetilde{S}/S = \left(\prod_{i=1}^{l-1} U_i^{a_i}\right)^B \bmod N \tag{6.6}$$

この時, ステップ 8 の判定は次の様に書ける.

$$\mathcal{O} = \begin{cases} OK & (\epsilon = \pm 1) \\ NG & (\epsilon \neq \pm 1) \end{cases} \tag{6.7}$$

なお, $B \in Z_N^*$なので, 新たに$\tilde{\epsilon}$を,

$$\tilde{\epsilon} = \prod_{i=1}^{l-1} U_i^{a_i} \bmod N \tag{6.8}$$

と定義して$\tilde{\epsilon}$をϵの代わりに用いても同じ判定結果になることに注意せよ. このことによってシミュレータは正しいプロトコルで使われている Bを知らなくてもシミュレートできる.

以上のことを踏まえて次のようなシミュレータを考え, その出力がもとのプロトコルの出力と多項式時間識別不能 (Polynomially indistinguishable) であることを示す.

[シミュレータ M_1]

(1) M, Nを S'に与える.

(2) $\boldsymbol{X}_1$を S'から得る.

(3) Z_N^*上のランダムな b 次元ベクトル $\boldsymbol{Y}_1$を生成し, ($\boldsymbol{Y}_1$の各要素を Z_Nから取り出し, 多項式時間でそれが Nと互いに素であることを確認すれば良い) S'に与える.

(4) S'から $\boldsymbol{X}_2$を得る.

(5) $\boldsymbol{Y}_2$を $\boldsymbol{Y}_1$と同様にして生成し, S'に与える.

(6) S'から Zを得る.

(7) $Z_{N'}$ (但し, $N' = (N-1)/2$) から奇数$\tilde{A}$を一様ランダムに選ぶ. ここで

$$\tilde{A} = \sum_{i=0}^{l-1} \tilde{a}_i b^i \tag{6.9}$$

と表されるものとする.

(8) $\widetilde{T} = \left(\sum_{i=1}^{l-1} X_{1,i}^{\tilde{a}_i}\right) \cdot M^{\tilde{a}_0} \bmod N$ を計算.

(9) $T = M^{\tilde{A}} \bmod N$を計算.

(10) $\tau = \widetilde{T}/T \bmod N$として, 次の規則で$\mathcal{O}$を決める.

$$\mathcal{O} = \begin{cases} OK & (\tau = \pm 1) \\ NG & (\tau \neq \pm 1) \end{cases} \tag{6.10}$$

(11) $\alpha = (N, M, \boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{X}_2, \boldsymbol{Y}_2, Z, \mathcal{O})$ を出力. (シミュレータ終)

この様に生成された系列α と正しいプロトコルの生成する系列の確率分布について考察する. 与えられた M, Nに対して, $\boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{X}_2, \boldsymbol{Y}_2, Z$ はそれぞれ, それ以前の系列に応じてシミュレータと正しいプロトコルで同一の確率分布で生成される. 確率を$\pi()$で表現してより正確に書き下せば,

$$\begin{aligned} &\pi(N, M, \boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{X}_2, \boldsymbol{Y}_2, Z, \mathcal{O}) \\ &= Prob(\mathcal{O}|N, M, \boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{X}_2, \boldsymbol{Y}_2, Z) \cdot Prob(Z|N, M, \boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{X}_2, \boldsymbol{Y}_2) \cdot \\ &\quad Prob(\boldsymbol{Y}_2|N, M, \boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{X}_2) \cdot Prob(\boldsymbol{X}_2|N, M, \boldsymbol{X}_1, \boldsymbol{Y}_1) \cdot \\ &\quad Prob(\boldsymbol{Y}_1|N, M, \boldsymbol{X}_1) \cdot Prob(\boldsymbol{X}_1|N, M) \cdot Prob(N, M) \end{aligned} \tag{6.11}$$

ここで右辺の 7 つの各項を左から順に $P_1, P_2, \cdots, P_7$とすると, P_3, P_4, P_6は S'によって定まり, P_7は与えられた分布, さらに, P_5, P_6は各々$1/|Z_N^*|^{b-1}$. よって, P_1のみがシミュレータと正しいプロトコルで異なる可能性がある. 与えられた M, N, $\boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{X}_2, \boldsymbol{Y}_2, Z$に対し, 正しいプロトコルの P_1は変数 A の値のみに, またシミュレータの P_1は変数$\tilde{A}$の値のみに依存する.

以上の考察をまとめると,αから$\mathcal{O}$を除いた系列をα^-と表す時,

$$\begin{aligned} \pi(\alpha) &= \pi(\alpha^- \mathcal{O}) && (6.12) \\ &= \pi(\alpha^-)\pi(\mathcal{O}|\alpha^-) && (6.13) \end{aligned}$$

ここで,$\pi(\alpha^-)$ は A や$\tilde{A}$とは独立であり, $\pi(\mathcal{O}|\alpha^-)$ のみが$\tilde{A}$に依存する.

今, $|\{A\}| = n_0$, $|\{\tilde{A}\}| = n_1$とすると, A, $\tilde{A}$は各々一様ランダムに選ばれるので, 特定の A, $\tilde{A}$が選ばれる確率はそれぞれ $1/n_0$, $1/n_1$である. 付録 E.1で示すように, Nが十分大きければこの二つの確率の差は無視できる程小さく, その結果観測される正しいプロトコルとシミュレータの生成する系列は多項式時間識別不能である. (証明終)

以上クラス 1 攻撃について証明したが, クラス 2 攻撃についても同様にシミュレータが構成できる (付録 E.2). なお, クライアントは結果$\widetilde{S}$を E乗してもとのメッセージに戻るか否かでOK/NGを決めていたが, シミュレータでは正しい Dを知らないのでこの手法は適用できない. シミュレータ自身が正しいべき乗 Tを計算してそれと依頼計算結果$\widetilde{T}$とが一致するか否かを検査して OK/NG を決めている.

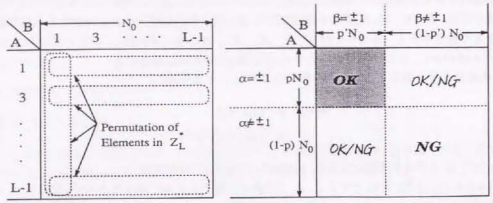
6.3.2 クラス 3 攻撃について

クラス 3 についてプロトコルが安全であることの証明はできていないので, 安全性を証明することはひとまずあきらめて, 代わりに秘密に関する情報がどのような仕組みで漏れる可能性があるかについて考察する. 計算量を度外視すれば何らかの手段で秘密が漏洩する可能性があることをを示すが, 将来計算量的に現実的な時間で秘密を取り出すことが困難であることが示せる可能性は残っていることに注意されたい.

クラス 3 攻撃を考察するために (A, B) と Dの関係について図 6.2に照らして考えよう. 図 6.2は以下に示す補題に基づいて描かれたもので, 表 (a) において, 行と列はそれぞれ A, Bの取り得る値に対応しており, i 行 j列には $D_{ij} = A_i \cdot B_j \bmod L$ が書かれている. A と Bは Z_L^*から選ばれているので, 表は $N_1 \times N_1$行列である.

補題 1 $\mathcal{O} = OK$の十分条件は$\alpha \in G_0$かつ$\beta \in G_0$. また, $\mathcal{O} = NG$ の十分条件はα, βの内, 一方のみが G_0の要素であることである. (証明は付録 E.3参照)□

図 6.2の表 (b) は (a) とほぼ等価な表であるが, 攻撃者が $Prob(\alpha \in G_0) = p$、$Prob(\beta \in G_0) = p'$ (但し $G_0 = \{1, N-1\}$) なる特性をもつ不正を行った場合を想定し, さらに行の入れ替え, 列の入れ替えを施して$\alpha = \pm 1, \beta = \pm 1$ に対応する領域が左上にくるようにしたものである. なお, A に関する OK確率を次の様に定義する.



(a) D = A·B mod L Table (b) Sorted Table

図 6.2: (A,B) と OK/NG 応答

定義 3 攻撃者がクラス 1 攻撃（逸脱 $\boldsymbol{U}$）をしたとする．この時，$Prob[\alpha \in G_0]$ を変数 A の $\boldsymbol{U}$に関する OK 確率と呼ぶ．確率は $\boldsymbol{U}$ を固定して，すべての A が等確率に取られるものとして定義する．□

$A, \alpha, \boldsymbol{U}$をそれぞれ $B, \beta, \boldsymbol{V}$に置き換えることにより，Bに関する OK 確率が同様に定義される．

表 (a) の特徴として行，列各々が Z_L^*の要素の並べ変えになっているという事に注意．すなわち，Z_L^*の任意の要素は各行、各列に丁度 1 度だけ現れる．従って，各行，列は正しい Dを必ず一つ含む．この特徴は任意の二つの行の入れ替え，任意の列の入れ替えに関して保存される．それ故，表 (b) も同じ特徴を有する．

表 (b) において与えられた逸脱 $\boldsymbol{U}$, $\boldsymbol{V}$ を一つ固定したときに，プロトコルで使われている D, およびクライアントの応答 $\mathcal{O}$ の関係について考える．プロトコルで使われている Dは常に固定であり，実際にプロトコルで使用される (A, B) のペアは丁度 N_1通り存在し，それらが等確率で選ばれる．またそれらの (A, B) は各行，各列に丁度一つずつ含まれる．

補題により表 (b) の 4 つに分けられた領域の内，$(\alpha, \beta) = (\pm 1, \pm 1)$ の領域に含まれる (A, B) が使われている時にはプロトコルの応答は OKである．それ以外は図に示すように NG のみが返される領域，NG/OKが確定できない領域 (但し，NG の確率が一般には高いと予想される) である．

プロトコルで使われている Dの値によっては OK 領域に片寄った (A, B) を持つものも有れば，別の Dでは NG 領域に (A, B) が片寄っていることもあろう．その片寄りかたは攻撃者によって選ばれた逸脱 $\boldsymbol{U}$, $\boldsymbol{V}$ に依存する．

極端なケースとして，すべての Dに対して同じ OK/NG 確率を有するような逸脱 $\boldsymbol{U}$, $\boldsymbol{V}$ を考えてみよう．明らかに，このような逸脱によってプロトコルで使われている Dを推定することは，不可能である．このことは攻撃に有効な逸脱は異なる Dまたは Dのグループに対して異なる OK確率を与えるものであることを示している．効率的にグループ分けできる逸脱の系列を生成することが計算

量的に可能で有ればクラス 3 攻撃でプロトコルは破れることになろう．なお，OK/NG 確率で Dを分類するには真の OK/NG 確率を観測によって推定しなければならないので，特定の逸脱について適当な推定信頼度を設定し何度か応答を観測する必要がある．前章のプロトコル 3 に対する逸脱が 1 度の問い合わせで Dをグループ分けできたのと比べ，この場合にはかなり攻撃の効率が下がるものと予想される．

以上，クラス 3 攻撃で Dに関する情報あるいは知識を取り出す手法について考察したが，実際効率的な攻撃が構成できるかは未解決問題である．

6.4 考察

6.4.1 処理性能

改良版プロトコル 5 とプロトコル 3 の計算量·通信量·メモリ量の比較を表 6.1に示す．プロトコル 5 はほぼプロトコル 3 を 2 回行うのと同じ内容を持つので，処理量も約 2 倍となっている．

表 6.1: 計算量の比較

Protocol	Computation*		Communi-cation†	Memory‡	
	Client	Server		Client	Server
3 (Basic)	$(1-1/b)l$	$(l-1)\log_2 b + 2b - 4$	$l+b+1$	$b+2$	$b+1$
5 (Improved)	$2(1-1/b)l$	$2(l-1)\log_2 b + 4b - 8$	$2(l+b)$	$2b+4$	$b+1$

Unit: * Modular multiplication, † m bit, ‡ m bit word
Note: Pre-computation and verification are neglected.

6.4.2 パラレル版について

プロトコル 3 の別な拡張法として，Dを 2 数の積では無く和として $D = (A + B) \bmod L$ と定めれば，並列版を考えることもできる．これを図 6.3に示すが，図中，(X, Y_A, Z_A) と (X, Y_B, Z_B) はそれぞれ秘密指数を A, Bとしてプロトコル 3 を実行した時に計算される値である．並列版において，不正ベクトル

$$X_1 = [X_1 U_1, X_2 U_2, \cdots, X_{l-1} U_{l-1}] \tag{6.14}$$

に対して計算される $\tilde{S}$は次のように書ける．

$$\tilde{S} = S \cdot \gamma \tag{6.15}$$

ここで，

$$\gamma = \prod_{i=1}^{l-1} U_i^{a_i + b_i} \tag{6.16}$$

プロトコル 5 と違い、$(a_i + b_i)$ は d_i と強い相関をもつので, 並列版の安全性はプロトコル 3 の安全性を飛躍的に増加させることはないと予想される. 詳細は検討を要する.

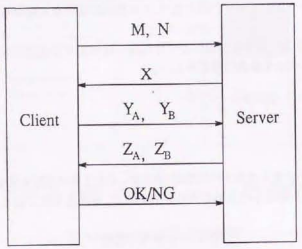


図 6.3: パラレル版プロトコル

6.5 むすび

この章ではべき乗の依頼計算において検算の結果をサーバに渡しても安全性を確保するために, 松本らが提案した 2 回依頼計算するというアイディアを, 我々自身が提案した基本プロトコルの拡張に利用し, その安全性について考察した. 改良版に対する攻撃を 3 通りに分類し (クラス 1, 2, 3), クラス 1, 2 攻撃に対しては改良版は安全であることが証明できたが, クラス 3 の安全性をどのように評価するかは今後の検討課題である.

第 7 章

システム応用 –暗号電子メール–

7.1 はじめに

電子メールは手軽で便利な通信手段として大学や研究機関を中心として急速に普及しつつある. しかし, 現在の多くの電子メールシステムは中継ノードでの内容漏洩防止機能やメッセージおよび送信者の正当性確認機能を持たず, 重要情報を電子メールで送信しようという場合の障害となっている[1].

ここでは新しい鍵配送プロトコルを提案すると共に, 前章までに検討してきた個別の検討結果を統合して高い安全性を実現できる電子メールシステムを開発したのでその概要を述べる. 以下ではまず, システム設計の基本方針を述べ, 次にセキュリティ・メカニズムについて説明し, 最後に UNIX 電子メールを利用した試作システムを紹介する.

7.2 基本方針

提案システムで実現するセキュリティ・サービスは次の 3 つである :

- メールの内容の第 3 者への漏洩防止
- 送信者の確認
- 改ざんの有無の確認

漏洩防止では, 特に通信回線・中継ノード・保存メールからの情報漏洩防止を目的とする. これらを実現する暗号メカニズムとしては、慣用暗号と公開鍵暗号を目的に応じて使い分けるハイブリッド方式を採用する. すなわちメール本体の暗号化には高速処理に向いた慣用暗号を用い, 慣用暗号で用いる鍵の配送には公開鍵暗号を用いる. 慣用暗号の鍵はメール毎に送信者が生成した乱数から定めら

[1] 近年インターネットでは電子メールの秘密保護・送信者確認・改ざんの検出機能を実現する PEM サービスが始まっている (PEM=Privacy Enhanced Mail, '93 年仕様が定められ, 日本では '94 年からサービスイン) 本章の研究はそれ以前に行われたものであり使用メカニズムも異なっている. なお, PEM と類似の機能を実現するシステムに PGP(Pretty Good Privacy) がある.

れ, この乱数を送信者が受信者の公開鍵を用いて暗号化して暗号メッセージに添えて受信者に送る. ここに述べたように提案方式は, 伝送遅延が比較的大きい電子メールという形態に適するよう受信者から送信者への通信を必要としないように設計されている.

改ざんの有無の確認と送信者の確認は, 送信者がメッセージに付加したディジタル署名によって同時に実現する.

7.3 暗号メカニズム

7.3.1 センタ処理

本システム全体を通して信頼のおけるセンタを仮定しており, 情報の認証は各個人に一意に割当てられた ID 情報 (識別情報=名前) に基づいておこなう. 信頼のおけるセンタはシステムの立ち上げとユーザの登録業務を行う.

以下ではユーザ i に対応する情報は添え字 i $(i = 1, \ldots, k)$ を付して示す. i にはシステム内で一意に定まる名前、ID_iが付いているものとし, プロトタイプシステムではメールのアドレスを ID_i として用いる. 本システムは次のような 3 種類の情報から構成される.

- センタ秘密情報： $P, Q, L = lcm(P-1, Q-1), E_i$
- 公開情報： ID_i, N, U, g, P_i, h
- ユーザ秘密情報： D_i, S_i

これらの内, ID_i以外はセンタが作成する. なお, 各記号の定義は以下の通りである.

N ：素数 P, Q の積でシステムに共通の法.

U ：L と互いに素な整数でセンタ署名確認用指数.

g ：システムに共通の離散対数の底.

D_i ：L と互いに素な整数で $i \neq j$ ならば $D_i \neq D_j$.

E_i ：$E_i \cdot D_i \equiv 1 (\mathrm{mod}\, L)$ を満たす整数.

h ：一方向性関数で, ユーザ i の ID 変換情報 I_iを $I_i = h(ID_i)$ と定義する. このとき S_i, P_iは次の関係を満たす整数である.

$$S_i^U = I_i^{-1} \bmod N \tag{7.1}$$

$$P_i = S_i \cdot g^{E_i} \bmod N \tag{7.2}$$

7.3.2 鍵配送プロトコル

ユーザ i がユーザ jに暗号メールを送信する場合の手順を説明する. なお, メッセージ Mの鍵 K による暗号化を $eK(M)$, 暗号文 Cの鍵 Kによる復号を $dK(M)$ と書く。

[ユーザ i の処理]

ステップ 1: ユーザ jの公開情報 P_jを公開鍵リストから入手し, 生成した乱数 r_iを用いて R, Xを計算する.

$$R = (P_j^U \cdot I_j)^{r_i} \bmod N \tag{7.3}$$

$$X = R^U \bmod N \tag{7.4}$$

ステップ 2: 鍵 $K = g^{r_i U^2} \bmod N$を作成する.

ステップ 3: 鍵 Kで平文 Mを暗号化する.

$$C = eK(M) \tag{7.5}$$

ステップ 4: Xと Cから $E = h(C, X)$ を求め, 暗号文 Cに対するユーザ i の拡張 Fiat-Shamir 署名 Yを計算する.

$$Y = S_i^E \cdot R \bmod N \tag{7.6}$$

ステップ 5: (ID_i, C, E, Y) をユーザ jに送信する.

[ユーザ jの処理]

ステップ 1: ID_i, E, Yから次式により X'を求める.

$$X' = Y^U \cdot I_i^E \bmod N \tag{7.7}$$

ステップ 2: 暗号文 Cと X'から $E' = h(C, X')$ を計算し, 受信した Eと E'が一致することを確認する. 一致しない場合には処理を中止する.

ステップ 3: 共有鍵 Kを次式により求める.

$$K = X'^{D_j} \bmod N \tag{7.8}$$

ステップ 4: 鍵 Kで暗号文を復号して Mを得る.

$$M = dK(C) \tag{7.9}$$

(終)

なお, 複数の受信者に共通の乱数 r_iを用いる形で上記方式を同報通信に利用することも検討してきたが [32], 異なる受信者に同一の r_iを使用した場合, 送信者秘密情報 S_iが漏れるので好ましくない. 同じ事は他のいくつかのグループ鍵共有方式に対しても言えることを [8], [6] で論じた. それ故, k 人の受信者への同報通信の場合, 2 者間のプロトコルを独立に k回繰り返さなければならない.

7.3.3 Fiat-Shamir 法との関係

先に述べたプロトコルでは, ユーザ i から送信される暗号文 Cに E,Yが依存しているため, 受信者のステップ 2 におけるチェックを通過する 4 つ組 (ID_i, C', E', Y') を偽造することは困難である. より正確に言えば, (E,Y) は暗号文 Cに対する送信者 i の拡張 Fiat-Shamir 法に基づくディジタル署名となっている. これによって本システムでは電文の完全なコピーを再送する場合を除けば, 第三者による暗号文の再利用を防止できる.

Fiat-Shamir 署名 [91] には文献 [92][93] [94] に示されているような幾つかの変形版が存在するが, 上記手続きの中で使用しているのは文献 [92][93] に示された版である. オリジナル版や他の版を用いても同様のプロトコルが構成できる. ただし, オリジナル版や Eの代わりに Xを署名の一部として用いる版では若干通信量が増すので, ここでは Eを用いる版を使っている.

上記手続きでは, オリジナルの Fiat-Shamir 署名の場合と同様に, Eは Cと Xから導かれており, 署名としての安全性は Fiat-Shamir 法の安全性と同等と考えることができる. 一方, そのような対応づけはできないものの, 実用上は $E = h(C,K)$ の様に Eを C,Kから導いても署名の安全性は劣化しないと予想される. この様にした場合には, 送信者が Rを Xから導く処理を省略できるので計算量をその分減らすことができる.

ここまでは Fiat-Shamir 認証法の内, 特に署名法 (メッセージ認証) をプロトコルに適用するという立場でいくつかの変形の可能性について論じてきた. 署名法は, 送信者から受信者への一方向の通信と整合が良いことがその主な理由である. しかし, 電子メール以外の通信形態で送信者受信者が会話的に通信する場合には, 署名法のみならず Fiat-Shamir の相手認証法 (identification protocol) も鍵共有に利用することができる. すなわち, 受信者 jが乱数 Eを生成し送信者 i に送るという構成である. またこの場合には, 受信者の公開情報 P_jを受信者から直接もらうことができるので, 公開情報ファイルを設ける必要もなくなる.

7.4 UNIX メールへの適用

7.3節に示したセキュリティサービスを UNIX メールで実現するシステム (ソフトウェアおよびハードウェア) を開発し, S メールシステムと名付けた. S メールシステムのポイントを列挙すると,

- セキュリティメカニズムとしては前節の鍵配送方式が中心となる.
- べき乗剰余演算には第 2 章で提案したアルゴリズムを適用.
- DSP で公開鍵暗号の演算を実現.
- 依頼計算の利用可能性を試算により検証.
- LAN 上で使い勝手を検証.

S メールは社内約 100 人のユーザが使用する LAN 上で試験運用し, 基本仕様を検証した. 試験運用の際の代表的端末は SUN-4 であったが, 端末構成は IC カードを用いるか否か, 用いる場合にはべき乗演算補助装置 (DSP による処理) を用いるか否かによって以下の 3 通りの場合にさらに分かれる.

構成 1 : 端末単独 (IC カードは用いない)

構成 2 : 端末+ IC カード・リーダ／ライタ

構成 3 : 端末+べき乗補助装置+ IC カード・リーダ／ライタ

リーダ／ライタは RS-232C インタフェースにより端末に接続される. 補助装置の主な構成部品は DSP であり, 現在は端末と IC カード・リーダ／ライタとの間に直列接続される構成を採っているが, 将来リーダ／ライタとの一体化が可能である. 各ユーザの公開鍵はユーザ共用ファイルに登録されており, 秘密鍵は個人のファイルか (構成 1 の場合), IC カード (構成 2, 3 の場合) のいずれかに格納される. 構成 2, 3 では IC カードは個人の秘密鍵の格納に用いるが, 構成 1 では IC カードを用いないので個人の秘密鍵は各自のファイルに格納する. S メールのユーザ・インタフェースは Nemacs のメール機能を採用しており, 暗号機能は Nemacs のコマンドの如く利用できる. S メールの主要諸元を表 7.1に示す. なお, 依頼計算は第 3 章の松本らの方式を用いた場合の見積り値を示しており, その実装は試作システムでは行っていない.

表 7.1: S メール主要緒元

代表的端末	SUN-4 相当品
ユーザ I/F	Nemacs/mail, rmail
送信処理時間	バックグランド処理のためユーザには知覚されず
受信処理時間	(7+m/10kbyte) 秒... 構成 1, 2 (1+m/10kbyte) 秒... 構成 3 (10+m/10kbyte) 秒... 構成 3 :依頼計算

m はメッセージバイト数. 依頼計算は試算.

7.5 むすび

電子メールシステムに適した秘匿・認証メカニズムを開発し, UNIX 電子メールに適用した結果について述べた. 本システムは情報漏洩防止・送信者認証・改ざん検出機能を有する. これらの機能を実現するために本章では拡張 Fiat-Shamir 署名を利用した鍵配送法を提案した. 提案システムは実験を通してその実用性を確認した. なお, システムが大規模化した場合のセンタ運営などの問題は未検証であるが, それらはこの章で示した基本構成に変更を加えることなしに解決する事が可能であると考える.

第 8 章

結論

8.1 本論文の成果

本研究は, 公開鍵暗号の実用化を目的とし, 特にべき剰余演算の高速処理のためのアルゴリズムを開発し, その一応用として電子メールの守秘・認証に応用した. 本論文の要点は次の様にまとめられる.

- テーブル参照による高速のべき乗剰余アルゴリズムを提案した.
- 松本らによって提案された依頼計算法の性能解析をした.
- べき乗計算のための依頼計算法を提案するとともにその安全性を厳密に解析した.
- 鍵配送方式を考案し守秘・認証機能を持つ電子メールを開発した.

以下本文の章だてに従って全体を要約する.

本論文は通信・情報システムのセキュリティを実現するために, RSA 暗号に代表される公開鍵暗号の実用化を目的として, 公開鍵暗号を実現するのに適用の広いべき剰余演算の高速化と, 依頼計算と呼ばれる 2 者間の計算分担プロトコルの設計, 処理性能, および安全性について検討すると共に, 秘匿や認証の機能のシステムへの実装を検証するために開発した暗号機能付き電子メールシステムについて論じたものであり 8 章からなる.

第 1 章では, 情報セキュリティ研究されている背景について述べるとともに研究の目的, 意義, 結果の概要を述べた.

第 2 章はテーブル参照による高速べき乗剰余計算について論じ, RSA 暗号, DH 鍵配送など整数論に基づく暗号で広く用いられているべき乗剰余演算の高速化について検討し, べき乗を乗算の繰り返しに分解した場合, その 1/3 は変数に定数を掛ける乗算であることを利用して高速化をはかる手法

を提案し, 提案方式の適用により従来方式よりも約 17%処理時間が短縮できることを示した.

第 3 章から第 6 章はべき乗計算のための依頼計算プロトコルについて論じたものである. 依頼計算とはクライアントと呼ばれる第 1 の装置が所持する秘密を, 外部に漏らすことなくサーバと呼ばれる第 2 の装置の補助を借りて, 単独で計算するよりも高速に行うためのプロトコルである.

第 3 章では, べき指数 Dの秘密を守りながらべき乗剰余演算を高速化するために松本等が提案した S2 プロトコルの性能解析を行い, 多様なサーバの性能, 種々の通信回線速度での処理時間削減効果を示した. ここで論じた S2 プロトコルは他の方式と比べて処理速度が速いという特長を有するものの, サーバに渡される依頼情報が秘密のべき指数と完全には独立でないために, 秘密指数がクライアント以外に漏れないという証明ができていないという問題を持っていた.

第 4 章では上に述べた S2 プロトコルの問題点を解決するために, 高速なべき乗計算アルゴリズム (Yao 法, Knuth 法) を (A) べき指数に依存する部分と, (B) べき指数に依存しない部分に分けて, (B) をサーバに行わせると言う方針で設計した依頼計算を 4 種類提案し, サーバがどのようにプロトコルから逸脱してもクライアントがプロトコルに従っている限り, クライアントから指数 D に関する新たな知識は漏れないこと (零知識性: Zero-knowledge) を証明した. ここで提案されたプロトコルと S2 プロトコルを比較した場合, 処理速度は S2 プロトコルより劣るものの, 安全性が証明付きで与えられているという顕著な特長を有する. 「対話型証明システム (Interactive Proof System)」の安全性を論ずる尺度として注目を集めている零知識性の概念を, 依頼計算という新たな問題に適用しその有効性を示した点にも新規性がある. Quisqater らはこれ以前に秘密指数 Dに関する情報が全く漏れていないと言う意味で著者らの提案したプロトコルと同じ安全性を有するプロトコルを提案していたが, 零知識性との関係については全く着目しておらず, プロトコルの性能も提案プロトコルより劣っている.

第 5 章では, 第 4 章とは別の前提条件で依頼計算プロトコルについて考察した. 第 4 章で提案した依頼計算プロトコルでサーバがプロトコルから逸脱した場合, 計算結果は逸脱の仕方と秘密指数 D に依存して正しい場合と正しく無い場合が生じる. 前章ではこの正誤情報はサーバに渡らないと仮定して零知識性が証明できたが, 正誤情報がサーバに渡る場合には零知識性は保証されない. この章ではクライアントがサーバに正誤情報を漏らした場合に, どのようなメカニズムで秘密指数 Dに関する情報が漏れるのかについて情報理論的尺度を用いて論じ, プロトコルの通信量が比較的大きいパラメータが選ばれた場合には, 秘密指数 Dをクライアントから取り出す具体的手順が存在することを示した. さらに, その攻撃法に対する現実的な対策を示し, その有効性を情報量の尺度を用いて示した.

第 6 章では, 第 5 章で論じたように正誤情報から秘密指数に関する情報が漏れることを考慮して, 正誤情報をサーバに漏らしても従来方式より安全性が保たれるプロトコルを提案する. 着目したのは秘密指数 Dを毎回ランダムに選ばれた二つの指数 A, Bの積の形に表すことによってプロトコルの安全性を高めようという方法で, この方法は松本らによって最初に提案され彼らの提案した S2 プロトコルの安全性を改善する為に用いられた. この章では第 5 章で論じたプロトコルに同じアイディアを

適用し, 証明を試みている. 改良版のプロトコルの安全性を全面的に証明はできなかったが, 攻撃法を 3 クラスに分類し, その内 2 つのクラスについては Dに関して零知識であるという証明を付けることができた. クラス 3 については証明を付けることはできなかったが, 元のプロトコルに比べ攻撃に対する抵抗力が増していると予想される理由を述べた.

第 7 章では, LAN 上で稼働している電子メールに守秘と認証機能を付加した暗号電子メールシステムについてプロトタイプを製作し, そこで用いられている Fiat-Shamir 法に基づいて新たに提案した鍵配送メカニズムを中心に, その構成を述べている. 鍵配送メカニズムにはべき乗計算が基本演算として用いられており, 第 2 章で提案したべき乗アルゴリズムを適用しており, 第 3 章で解析した依頼計算を用いた場合の性能を検討した.

8.2 今後の課題と展望

以上, 章の流れに従って本論文の主たる成果を述べてきた. 本論文の範囲で最も大きい未解決課題は, べき乗の依頼計算において計算の正誤結果を漏らしても証明付きで安全な方式の開発が挙げられる. 本論文では秘密のべき指数を二つの変数の積に分割して, 基本となるべき乗演算プロトコルを 2 回走らせると言うアイディアに基づいて改良方式を構成したが, 安全性の全面的証明はできなかった. 解決の方向としては提案プロトコルについて証明手法を考案すること, それが困難ならば, 別の発想による証明付きのプロトコルを提案することが挙げられる. また, 仮に提案プロトコルの安全性が証明できた場合にも, 提案プロトコル処理効率は基本プロトコルよりも落ちている. 結果の正誤を漏らしても安全性を保て, しかも処理速度が高速な方式を考案することはいずれにしても今後の課題である.

依頼計算については 1987 年の概念の発表以来主に日本を中心に RSA 暗号への応用が研究されてきたが, 近年ディジタル署名のチェックなど新たな分野開拓を含む研究がヨーロッパ、アジアでも行われつつある [100][101][102][103]. こうした研究の流れについては [12] にまとめられている.

さらに広く暗号技術という観点で今後の研究の方向を展望すると, 学術的波及効果の大きさから暗号を含むセキュリティ技術の安全性評価技術が, 応用システムを求める社会的ニーズからは電子現金・電子投票と言ったネットワークを介して実現される安全なマルチパーティプロトコルの研究が重要であろう. 安全性の評価技術については零知識性に代表される計算量理論的アプローチ, 実用上重要な慣用暗号の安全性の評価の発展が予想される. 応用システムの開発, 研究もまだその緒についたばかりであり一層深みのある研究が求められている.

近年, 通信・情報機器の小型化, 広域化, 社会資本化に一層拍車が掛かっており, 暗号技術に限らずセキュリティ技術の重要性が増していることは疑いない. しかし, 冒頭で述べたようにセキュリティは技術のみによって確保できるものではない. 法律や制度, ひいては人間性やモラルにまで関わっている. 今後そうした技術以外の領域の要求と, 技術との融合をどうはかっていくかと言う点にも工学

者が目を向けて行かねばならない時期が来ていると思われる. 本論文の検討がセキュリティ研究ひいては通信工学の発展にいくらかでも役立つならば, 著者としてこの上ない喜びである.

謝辞

本研究をまとめるに当たり, 懇切なる御指導と励ましの言葉を頂いた東京大学工学部 電子情報工学科 羽鳥光俊教授に深く感謝いたします.

また有益な御助言と御指導を賜った東京大学 今井秀樹教授, 原島 博教授, 浅野正一郎教授, 相澤清晴助教授に厚く御礼申し上げます.

本研究の遂行に当たって, 理解ある御指導, 御配慮を賜った株式会社 東芝 笠見昭信 取締役, 同 コンセプトエンジニアリング開発部 南 正名 部長, に深く感謝致します.

また本研究遂行中, 広範囲に亘り具体的な御助言, 御指導を頂いた株式会社 東芝 福田武郎 統括技師長, 同 情報通信システム技術研究所 鈴木秀夫 所長, 同 情報通信システム研究所 杉山文夫 所長, 同 児玉利一 技監, 同 セキュリティ技術センター 才所 敏明 部長, 同 関西研究所 中村 誠 主幹, 同 半導体事業部 菅原 勉 主幹, 東芝を退社された浅川 繁博士, 東芝リサーチコンサルティング 村上純造博士, に深甚な謝意を表します.

さらに研究内容について指導, 討論, 実験をして頂いた株式会社 東芝 情報通信システム研究所 神竹孝至 部長, 新保 淳氏, 同 小向工場 阿部雅宏氏, 既に退社された高林京子氏, 富山大学 田島正登教授, に深く感謝いたします.

また共に仕事をしている東芝、セキュリティ技術センターの皆様に感謝いたします。

最後に本研究をまとめるに至るまで暖かい励ましの言葉を贈ってくれた父と妻に感謝します.

参考文献

以下 [1]–[53] は，著者の著作物で，それ以降は著者以外による文献．[14],[19],[36] は招待講演．[34] により電子情報通信学会学術奨励賞授賞．

論文

[1] Shin-ichi Kawamura and Atsushi Shimob: "Performance Analysis of Server-Aided Secret Computaion Protocols for the RSA Cryptosystem", *IEICE Trans.*, Vol.E73, No.7, pp.1073-1080, July 1990.

[2] Shin-ichi Kawamura, Kyoko Takabayashi and Atsushi Shimbo: "A Fast Modular Exponentiation Algorithm", *IEICE Trans.*, Vol.E74, No.8, pp.2136-2142, August 1991.

[3] Shin-ichi Kawamura and Atsushi Shimbo: "Fast Server-Aided Secret Computation Protocols for Modular Exponentiation", *IEEE JSAC*, Vol.11, No.5, pp.778-784, June 1993.

[4] Shin-ichi Kawamura: "Information Leakage Measurement in A Distributed Computation Protocol", *IEICE Trans.*, Vol.E78-A, No.1, pp.59-66, January 1995.

レター

[5] Atsushi Shimbo and Shin-ichi Kawamura: "Factorization Attack on Certain Server-Aided Computation Protocols for the RSA Secret Transformation", *Electronics Letters*, Vol.26, No.17, pp.1387-1388, August 1990.

[6] 新保 淳、川村 信一: "Fiat-Shamir 認証法を利用したグループ鍵共有方式の解析", 電子情報通信学会論文誌 A、Vol.J75-A, No.8, pp.1312-1316, 1992

国際学会／海外開催学会 (いずれも査読有り)

[7] Shin-ichi Kawamura and Kyoko Hirano: "A Fast Modular Arithmetic Algorithm Using a Residue Table", *Proc. of EUROCRYPT'88, Lecture Notes in Compuater Science*, No.330, Springer-Verlar, 1988.

[8] Atsushi Shimbo and Shin-ichi Kawamura: "Cryptanalysis of Several Conference Key Distribution Schemes", *Proc. of ASIACRYPT'91, Lecture Notes in Computer Science*, No.739, Springer-Verlag, 1991.

[9] Shin-ichi Kawamura and Kyoko Hirano: "A Fast Modular Exponentiation Algorithm Using A Residue Table", *Internationa Symposium on Information Theory*, Kobe (1988)

[10] Shin-ichi Kawamura and Atsushi Shimbo : "Intra- and Inter- Network Security", *TELECOM 91*, Geneva (1991)

[11] Shin-ichi Kawamura: "Information Leakage in A Distributed Compuation Protocol", *Proc. of Information Sciences and Systems*, at Princeton Univ., March 16-18,1994.

著書 (分担執筆)

[12] 川村 信一: 第?章 "零知識証明と依頼計算", 情報理論とその応用レクチャーノート, 培風館, 1996 出版予定

電子情報通信学会／情報処理学会研究会

[13] 川村 信一, 神竹 孝至, 水谷 博之: "ＩＣカードを用いた取引証明法", 情報処理学会研究報告, Vol.86, No.51, 1986

[14] 広川 勝久, 川村 信一: "ＩＣカード・システムにおけるセキュリティについて" (招待講演), 電子情報通信学会, 研究会資料 IT87-82, pp.15-25, 1987

[15] 川村 信一, 青山 光伸: "ＩＣカードのセキュリティ機能を利用したオフィスシステム", 電子情報通信学会, 研究会資料 OS87-50, pp.39-44, 1988

[16] 川村 信一, 新保 淳: "予備判定による素数生成効率化の解析", 電子情報通信学会, 研究会資料 ISEC88-18, pp.49-53, 1988

[17] 新保 淳, 川村 信一: "ＲＳＡ暗号の依頼計算における検算問題について", 電子情報通信学会, 研究会資料 ISEC89-5, pp.29-35, 1989

[18] 川村 信一, 新保 淳: "ＲＳＡ暗号の依頼計算における検算問題について (II)", 電子情報通信学会, 研究会資料 ISEC89-17, pp.29-34, 1989

[19] 川村 信一: "ＺＫＩＰと依頼計算" (招待講演), 電子情報通信学会, 研究会資料 ISEC89-59, pp.81-90, 1990

[20] 川村 信一, 新保 淳: "公開鍵暗号文の復号能力に基づく暗号方式について", 電子情報通信学会, 研究会資料 ISEC90-5, pp.29-34, 1990

[21] 新保 淳, 川村 信一: "Fiat-Shamir 署名に基づく一方向鍵共有方式の安全性について", 電子情報通信学会, 研究会資料 CS91-17 (RCS91-7), pp.1-6, 1991

[22] 新保 淳, 川村 信一: "「n 変数べき乗剰余演算とその応用」に関する考察" 電子情報通信学会, 研究会資料 ISEC91-59, pp.27-34, 1992

[23] Shin-ichi Kawamura: "A Server-Aided RSA Computation Protocol in Two Rounds", 電子情報通信学会, 研究会資料 ISEC94-05, pp.61-68, 1994

電子情報通信学会全国大会

[24] 川村 信一, 新保 淳: "物理的安全性に基づいた暗号化鍵共有法", 電子情報通信学会春季全大, A-288, p.1-290,1988

[25] 川村 信一, 新保 淳: "Rabin 暗号の署名を用いた鍵共有方式", 電子情報通信学会春季全大, A-289, p.1-291, 1988

[26] 新保 淳, 川村 信一: "試行割算法を組み込んだ素数判定法の解析", 電子情報通信学会秋季全大, SA-7-1, p.A-1-165, 1988

[27] 川村 信一, 新保 淳: "ＩＣカードシステムにおける依頼計算について", 電子情報通信学会春季全大, AS-8-3, p.1-411, 1989

[28] 川村 信一, 新保 淳: "ＲＳＡ暗号の依頼計算方式の拡張と処理時間の解析", 電子情報通信学会秋季全大, A-105, p.1-108, 1989

[29] 高林 京子, 川村 信一, 新保 淳: "定数乗算テーブルを用いたべき乗剰余計算法", 電子情報通信学会春季大会, A-285, p.1-285, 1990

[30] 新保 淳, 川村 信一: "ＲＳＡ暗号の依頼計算における素因数分解攻撃と対策", 電子情報通信学会春季大会, A-286, p.1-286, 1990

[31] 川村 信一, 新保 淳: "公開された暗号文の復号能力に基づく暗号方式", 電子情報通信学会秋季大会, A-169, p.1-172, 1990

[32] 川村　信一, 新保　淳: “Fiat-Shamir 署名方式を利用した一方向鍵共有方式”, 電子情報通信学会春季大会, A-292, p.1-294, 1991

[33] 新保　淳, 川村　信一: “グループ鍵共有方式に対する攻撃法”, 電子情報通信学会秋季大会, A-146, p.1-147, 1991

[34] 川村　信一, 新保　淳, 高林　京子: “セキュア電子メールシステム”, 電子情報通信学会春季大会, SA-4-2, p.1-426, 1992

[35] 川村　信一, 新保　淳: “零知識依頼計算プロトコル”, 電子情報通信学会秋季大会, A-187, p.1-190, 1992

電気情報関連学会連合大会

[36] 川村 信一: “キャッシュレス社会とＩＣカード”（招待講演）, 電気・情報関連学会連合大会, 東京電気大, 1991

情報理論とその応用シンポジウム

[37] Takashi Kamitake and Shin-ichi Kawamura: “On-Line Transaction Certification Suitable for IC Card Systems”, *10th Symposium on Information Theory and Its Application (SITA '87)*, pp.23-28, Enoshima, Japan, 1987.

[38] 新保　淳, 川村　信一: “試行制算法を組み込んだ素数生成法の解析”, 第１１回情報理論とその応用シンポジウム, pp.277-282, 別府, 1988

[39] 新保　淳, 川村　信一: “依頼計算の検算性能について”, 第１２回情報理論とその応用シンポジウム, pp.297-302, 犬山, 1989

[40] Shin-ichi Kawamura, Atsushi Shimbo and Kyoko Takabayashi: “E-mail implementation of a one-way key distribution scheme”, *14th Symposium on Information Theory and Its Application (SITA '91)*, pp.17-20, Ibusuki, Japan, 1991.

[41] 新保　淳, 川村　信一: “秘密情報に依存しないＲＳＡ暗号の依頼計算方式”, 第１５回情報理論とその応用シンポジウム, pp.269-272, 水上, 1992

暗号と情報セキュリティシンポジウム

[42] 神竹 孝至, 川村 信一: “IC カードとセキュリティ”, 暗号と情報セキュリティワークショップ講演論文集, pp.3-5, 1986

[43] 新保 淳, 川村 信一: “秘密情報を漏らさずに認証する法方について”, 暗号と情報セキュリティシンポジウム, 函南, 1988

[44] 川村 信一: “同一テーブルを繰り返し用いることによる剰余テーブルの削減”, 暗号と情報セキュリティシンポジウム, 函南, 1988

[45] 川村 信一, 新保 淳: “依頼計算による RSA 暗号の秘密変換”, 暗号と情報セキュリティシンポジウム, 御殿場, 1989

[46] 川村 信一, 新保 淳: “RSA 暗号に対する依頼計算法の処理時間解析”, 暗号と情報セキュリティシンポジウム, 日本平, 1990

[47] 新保 淳, 川村 信一: “RSA 暗号の依頼計算を悪用した素因数分解法と対策”, 暗号と情報セキュリティシンポジウム, 日本平, 1990

[48] 新保 淳, 川村 信一: “メッセージスタンプを用いた一方向型鍵共有方式”, 暗号と情報セキュリティシンポジウム, 富士吉田, 1991

[49] 川村 信一, 高林京子, 新保 淳: “多倍長べき乗剰余演算の高速アルゴリズム”, 暗号と情報セキュリティシンポジウム, 富士吉田, 1991

[50] 川村 信一, 新保 淳: “加算連鎖に基づく依頼計算法”, 暗号と情報セキュリティシンポジウム, たて科, 1992

雑誌寄稿

[51] 鈴木　秀夫, 川村　信一: “通信・情報セキュリティ技術”, 東芝レビュー, Vol.47, No.6, pp.472-473, 1992

[52] 川村　信一, 新保　淳, 高林　京子: “公開鍵暗号利用セキュリティ技術”, 東芝レビュー, Vol.47, No.6, pp.395-498, 1992

[53] 川村　信一: “ＩＣカードで秘密をどう守るか？”, エレクトロニクス, Vol.36, No.7, pp.52-55, オーム社, 1991

[54] 川村　信一: “依頼計算って何ですか？”, エレクトロニクス, Vol.41, No.5, pp.85-86, オーム社, 1996

その他の参考文献

[55] C. E. Shannon: “Communication Theory of Secrecy Systems”, *Bell Syst. Tch. J.*, 28, pp.656-715, 1949

[56] W. Diffie and M. E. Hellman: "New Directions in Cryptography", *IEEE Trans. Inform. Theory*, IT-22, 6, pp.644-645, Nov. 1976

[57] R. L. Rivest and A. Shamir and L. Adleman: "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Commun. of ACM*, pp.120-126, Feb. 1978

[58] D. E. Knuth: *The Art of Computer Programming, Vol2, Seminumerical Algorithms*, 2nd Ed., Addison-Wesley, 1976

[59] R. Ash: *Information Theory*, John-Wiley and Sons, New York, 1965

[60] A. C. Yao: "On The Evaluation of Powers", *SIAM J. Comput.*, Vol.5, No.1, pp.100-103, 1976

[61] M. Coster: "Some Algorithms on Addition Chains and Their Complexity", *Report CS-R9024*, Center Math. Comput. Sci.(CWI)", 1990

[62] 鳥居　直哉, 東　充宏, 秋山　良太: "RSA 分割計算処理の一検討", 1986 暗号と情報セキュリティワークショップ講演論文集, pp.15-17, 1986

[63] 永井　康彦, 宝木　和夫, 中川　聡夫, 佐々木　良一: "電子取引認証システムの機能試作機の開発", 1987 暗号と情報セキュリティワークショップ講演論文集, pp.109-121, 1987

[64] 加納　康男, 松崎　なつめ, 飴林　誠: "高次の拡張 Booth アルゴリズムを用いたべき乗剰余演算 LSI", 1987 暗号と情報セキュリティワークショップ講演論文集, pp.133-142, 1987

[65] 田中　和恵, 岡本　栄司: "シグナルプロセッサを用いた巾乗剰余演算について", 昭和 62 年電子情報通信学会情報・システム部門全大, p.15, Nov. 1987

[66] E. F. Brickell: "A survey of hardware implementations of RSA", *Advances in Cryptology-CRYPTO 88, Lecture Notes in Computer Science*, 435, pp.368-370, New York: Springer-Verlag, 1989

[67] P. Barrett: "Implementing The Rivest Shamir Adleman Public Key Encryption Algorithm On A Standard Digital Signal Processor", *Advances in Cryptology-CRYPTO 86, Lecture Notes in Computer Science*, 263, pp. 311-323, New York: Springer-Verlag, 1986

[68] P. L. Montgomery: "Modular Multiplication Without Trial Division", *Mathematics of Computation*, 44, 170, pp.519-521, 1985

[69] S. R. Dusse and B. S. Kaliski, Jr: "A Cryptographic Library For The Motorola DSP 56000 (Extended Abstract)", *Advances in Cryptology-EUROCRYPT'90, Lecture Notes in Computer Science*, 473, pp.230-244, New York: Springer-Verlag, 1991

[70] P. A. Findlay and B. A. Johnson: "Modular Exponentiation Using Recursive Sum of Residues", *Advances in Cryptology-CRYPTO 89, Lecture Notes in Computer Science*, 435, pp.371-376, New York: Springer-Verlag, 1989

[71] J. Bos and M. Coster: "Addition Chain Heuristics", *Advances in Cryptology-CRYPTO 89, Lecture Notes in Computer Science*, 435, pp.400-407, New York: Springer-Verlag, 1989

[72] Y. Yacobi: "Exponentiating Faster with Addition Chains", *Advances in Cryptology-EUROCRYPTO'90, Lecture Notes in Computer Science*, 473, pp.222-229, New York: Springer-Verlag, 1990

[73] 松本　勉, 加藤　光幾, 今井　秀樹: "秘密を漏らさずに IC カードが端末を用いて計算するには", 第 10 回　情報理論とその応用シンポジウム予稿集, 江ノ島, pp.17-22, 1987

[74] 松本　勉, 今井　秀樹: "秘密を漏らさずに計算力を借りる依頼計算法", 電子情報通信学会, 研究会資料, ISEC88-9, pp.53-59, 1988

[75] T. Matsumoto and K. Kato and H. Imai: "Speeding up Secret Computations with Insecure Auxiliary Devices", *Advances in Cryptology-CRYPTO'88, Lecture Notes in Computer Science*, 403, pp.497-506, New York: Springer-Verlag, 1990

[76] 松本　勉, 今井　秀樹: "検算可能な依頼計算", 電子情報通信学会, 研究会資料, ISEC89-4, pp.21-28, 1989

[77] J. Feigenbaum: "Encrypting Problem Instances, or ..., Can you take advantage of someone without having to trust him?", *Advances in Cryptology-CRYPTO'85, Lecture Notes in Computer Science*, 219, pp.477-488, New York: Springer-Verlag, 1986

[78] M. Abadi and J. Feigenbaum and J. Kilian: "On Hiding Information from An Oracle", *Journal of Comput. Syst. Sci.*, 39, pp.21-50, 1989

[79] D. Beaver and J. Feigenbaum: "Hiding Instances in Multioracle Queries", *Proc. 7th STACS, Lecture Notes in Computer Science*, 415, pp.37-48, New York: Springer-Verlag, 1990

[80] J. Feigenbaum and R. Ostrovsky: "A Note on One-Prover, Instance-Hiding Zero-Knowledge Proof Systems", *Advances in Cryptology-ASIACRYPT'91, Lecture Notes in Computer Science*, 739, pp. 352-359, New York: Springer-Verlag, 1993

[81] C. Laih and S. Yen and L. Harn: "Two Efficient Server-Aided Secret Computation Protocols Based on The Addition Sequence", *Advances in Cryptology-ASIACRYPT'91, Lecture Notes in Computer Science*, 739, pp.450-459, New York: Springer-Verlag, 1993

[82] J.-J. Quisquater and M. de Soete: "Speeding Up Smart Card RSA Computation with Insecure Coprocessors", *Proc. SMARTCARD 2000*, Amsterdam, pp.191-197, North-Holland, 1989

[83] S. Goldwasser and S. Micali and C. Rackoff: "The Knowledge Complexity of Interactive Proof Systems", *Proc. 17th Annu. ACM Symp. Theory Comput.*, pp.291-304, May, 1978

[84] B. Pitzmann and M. Waidner: "Attacks on Protocols for Server-Aided RSA Computation", *Advances in Cryptology-EUROCRYPT'92, Lecture Notes in Computer Science*, 658, pp.153-162, New York: Springer-Verlag, 1992

[85] J.-J. Quisquater and C. Couvreuer: "Fast Decipherment Algorithm for RSA Public-Key Cryptosystem", *Electron. Lett.*, Vol.18, No.21, pp.905-907, 1982

[86] 松本　勉, 今井　秀樹: "依頼計算について", 電子情報通信学会, 研究会資料, ISEC92-15, pp.7-10, 1992

[87] 松本　勉, 今井　秀樹: "依頼計算について（その２）", 電子情報通信学会, 研究会資料, ISEC92-54, pp.15-20, 1992

[88] 松本　勉, 今井　秀樹: "RSA 暗号の依頼計算プロトコルの現状", 暗号と情報セキュリティシンポジウム, 1993

[89] T. Matsumoto, H. Imai, C. S. Laih and S. M. Yen: "On Verifiable Implicit Asking Protocols for RSA Computation", *Advances in Cryptology-AUSCRYPT'92, Lecture Notes in Computer Science*, 718, pp.296-307, New York: Springer-Verlag, 1992

[90] S. M. Yen and C. S. Laih: "More About The Active Attack on The Server Aided Secret Computation Protocol", *Electronics Letters*, Vol.28, No.24, p.2250, 1992

[91] A. Fiat and A. Shamir: "How to Prove Yourself: Practical Solutions to Identification and Signature Problems", *Advances in Cryptology-CRYPTO 86, Lecture Notes in Computer Science*, 263, pp.186-194, New York: Springer-Verlag, 1987

[92] L. C. Guillou and J. J. Quisquater: "A Practical Zero-Knowledge Protocol Fitted To Security Microprocessor Minimizing Both Transmission And Memory", *Advances in Cryptology-EUROCRYPT'88, Lecture Notes in Computer Science*, 930, pp.123-128, New York: Springer-Verlag, 1988

[93] K. Ohta and T. Okamoto: "A Modification of The Fiat-Shamir Scheme", *Advances in Cryptology-CRYPT 88, Lecture Notes in Computer Science*, 403, pp.232-243, New York: Springer-Verlag, 1988

[94] K. Ohta: "Efficient Identification and Signature Schemes", *Electron. Lett.*, Vol.24, pp.115-116,

[95] T. Okamoto and K. Ohta: "How to Utilize The Randomness of Zero-Knowledge Proofs", *Advances in Cryptology-CRYPT 90, Lecture Notes in Computer Science*, 537, pp.456-475, New York: Springer-Verlag, 1990

[96] 田中　和恵, 岡本　栄司: "ID 関連情報ファイルを用いた一方向鍵配送方式", 電子情報通信学会, 研究会資料, ISEC89-3, pp.17-20, 1989

[97] T. Itoh, T. Habutsu, I. Sasase and S. Mori: "One-way Key Distribution System Based On Identification Information Withnout Public Information Directory (in Japanese)", 電子情報通信学会, 研究会資料, ISEC90, pp.87-90, 1990

[98] T. Chikazawa and T. Inoue: "A New Key Sharing System for Global Telecommunications", *Proc. of IEEE GLOBECOM'90*, pp.1096-1272, 1990

[99] 松崎　なつめ, 原田　俊治, 館林　誠: "公開鍵生成法の提案と暗号鍵配送方式の応用", 電子情報通信学会, 研究会資料, ISEC90-43, pp.37-46, 1990

[100] P. Beguin and J.-J. Quisquater: "Secure Acceleration of DSS Signatures Using Insecure Server", *Advances in Cryptology-ASIACRYPT 94, Lecture Notes in Computer Science*, 917, pp.249-259, New York: Springer-Verlag, 1994

[101] C.-H. Lim and P.-J. Lee: "Server (prover / signer)-Aided Verification of Identity Proofs and Signatures", *Advances in Cryptology-EUROCRYPT 95, Lecture Notes in Computer Science*, 921, pp.64-77, New York: Springer-Verlag, 1995

[102] P. Beguin and J.-J. Quisquater: "Fast Server-Aided RSA Signatures Secure Against Active Attacks", *Advances in Cryptology-CRYPT 95, Lecture Notes in Computer Science*, 963, pp.57-69, New York: Springer-Verlag, 1995

[103] C.-H. Lim and P.-J. Lee: "Security and Performance of Server-Aided RSA Computaion Protocols", *Advances in Cryptology-CRYPT 95, Lecture Notes in Computer Science*, 963, pp.70-83, New York: Springer-Verlag, 1995

付 録 A

第 1 章の付録

A.1 公開鍵暗号系と RSA 方式

暗号方式は慣用暗号と公開鍵暗号に大きく分けられる．慣用暗号は暗号化と復号に同じ鍵を用いる方式で，米国の DES(Data Encription Standard) が代表例である．一方，公開鍵暗号は暗号化と復号の鍵が異なり，しかも暗号化の鍵から復号の鍵が導けない方式であり，暗号化の鍵を公開しても安全性が保たれる．例えば，システムの各ユーザに公開鍵（暗号化の鍵）と秘密鍵（復号の鍵）をペアで発行し，公開鍵は公開ファイルにまとめて電話帳のように発行してある場合を考える．A 氏に暗号文を送り付けたければ，A 氏の公開鍵を用いて暗号文を作成して送る．これを復号できるのは対応する秘密鍵を持つ A 氏のみである．また公開鍵暗号はディジタル署名としてメッセージの作成者を認証する方式として用いられることもある．RSA 暗号は公開鍵暗号の一種で，暗号通信にもディジタル署名にも用いることができる優れた方式である．RSA 暗号の構成と典型的鍵の作り方を説明する：まず，大きい素数 P, Q を選び $N = PQ$, $L = lcm(P-1, Q-1)$ (lcm：最小公倍数) とする．次に，L と互いに素な L 以下の整数 Eを選び，$E \cdot D \bmod L = 1$ となる Dを定める。

- 公開鍵: E, N
- 秘密鍵: D, P, Q
- 暗号化: $C = M^E \bmod N$
- 復　号: $M = C^D \bmod N$

RSA 暗号の主な安全性の根拠は，P, Q からその積 Nを計算することは容易であるが，P, Q が充分に大きい素数の時，Nを与えられてそれを素因数分解するのは困難であることである．Nの素因数分解が困難であるためには，Nを十進数数百桁に選ぶ必要がある．

付 録 B

第 3 章の付録

B.1 $N(l,n)$ の導出

まず, 秘密のパラメータのベクトル $\boldsymbol{F}$, $\boldsymbol{G}$ が何通り存在するのか考察する. クライアントに許されている剰余乗算の回数は n 回なので, $(\boldsymbol{F},\boldsymbol{G})$ を集合の形で表すと,

$$\left\{(\boldsymbol{F},\boldsymbol{G})\left|\sum_{i=1}^{i}\chi(f_i)+\sum_{i=1}^{i}\chi(g_i)=n;\quad f_i,g_i \text{ は非負整数}\right.\right\} \tag{B.1}$$

ここで$\chi(x)$ は二進法を用いて x 乗を計算するのに必要となる乗算回数を表す. x を二進数展開したときのビット数とハミング重みをそれぞれ k,wとすると, $\chi(x)$ は次のように表せる.

$$\chi(x)=k+w-2 \tag{B.2}$$

つぎに, f_i, g_iに割り付けられた剰余乗算回数をそれぞれ n_i, n_{i+l}として, その $2l$値を要素として持つようなベクトルを u とし, 可能な u 全てからなる集合を $L=\{u|u=\{n_1,\cdots,n_{2l}\}\}$ とする. L を用いると, 集合 $(\boldsymbol{F},\ \boldsymbol{G})$ の要素の数は次のように書ける.

$$\#\{(\boldsymbol{F},\boldsymbol{G})\}=\sum_{u\in L}\left(\prod_{i=1}^{2l}\rho(n_i)\right) \tag{B.3}$$

ここここでρは 3.3.4節の本文中で定義されている.

さらに, $N(l,n)$ を導くには次の 2 条件を考慮した.

(a) $(P-1)$ と $(Q-1)$ の最大公約数を gとするとき, $\boldsymbol{D}\cdot\boldsymbol{F}$と$\boldsymbol{D}\cdot\boldsymbol{G}$は gを法として合同で無ければならないので, $\{(\boldsymbol{F},\boldsymbol{G})\}$ の要素の内, 秘密ベクトルとして使えるものの数は$\#\{\boldsymbol{F},\boldsymbol{G})\}/g$である.

(b) 付録 B.2に示す様に攻撃者から見たとき $(\boldsymbol{F},\boldsymbol{G})$ と $(\boldsymbol{G},\boldsymbol{F})$ は同じ Dの推定値を与えるので、$(\boldsymbol{F},\boldsymbol{G})$ の候補の内, 半数を探索すれば良い.

この 2 条件を考慮すると,

$$N(l,n) = \#\{(\boldsymbol{F},\boldsymbol{G})\}/2g = \sum_{u\in L}\left(\prod_{\nu=1}^{2i}\rho(n_i)\right)/2g \tag{B.4}$$

なお, gは最小値 2 に選ぶ事ができる.

B.2 秘密指数 D の推定

安全性を考察するに当たっては, 秘密ベクトル$\boldsymbol{F}$, $\boldsymbol{G}$の候補を総当たり的に探索して秘密指数 D を推定する次のような攻撃法を前提とした.

まず, ベクトル$\boldsymbol{F}$,$\boldsymbol{G}$および Dは次の関係を満たすように選ばれている.

$$D \equiv \boldsymbol{D}\cdot\boldsymbol{F} \bmod (P-1) \tag{B.5}$$

$$D \equiv \boldsymbol{D}\cdot\boldsymbol{G} \bmod (Q-1) \tag{B.6}$$

$$\tag{B.7}$$

次に, δを$\boldsymbol{F}$, $\boldsymbol{G}$の推定値から導かれる Dの候補とする.

$$\delta = \boldsymbol{D}\cdot(\boldsymbol{F}+\boldsymbol{G}) - E\cdot(\boldsymbol{D}\cdot\boldsymbol{F})(\boldsymbol{D}\cdot\boldsymbol{G}) \tag{B.8}$$

もし, $\boldsymbol{F}$, $\boldsymbol{G}$が正しく推定されれば, δは Dと等価である, すなわち

$$M^D = M^\delta \bmod N \tag{B.9}$$

が成り立つことが容易に示せる.

B.3 QS 法の性能

3.4節では QS 法の解析結果もグラフに示した. 我々がどのように QS 法を解析したかを示しておく. 使用したパラメータは次の通り.

b: 指数 Dを現すのに用いた底. Dは b 進数表現されている.

r: QS 法で用いられている, 特殊な基底を構成する要素の数.

k: b 未満の正整数を和の形で表現するのに必要な基底要素の数. b 未満の正整数は高々k個の基底要素の和として表現できる様に基底が選ばれている.

m: QS 法を実施した場合のラウンド回数.

クライアントの計算は中国剰余定理を利用した方法を用いるものとする. 正規化されたサーバの計算時間, 正規化されたクライアントの計算時間, 通信コストはそれぞれ次のように求められる.

$$T_S/T^Q = \{8\chi(b)(r+1)m/3\nu\}(1/v) \tag{B.10}$$

$$T_C/T^Q = 2km/\nu \tag{B.11}$$

$$T_{com} = m(r+1)+2 \tag{B.12}$$

文献 [82] で用いられているパラメータ $b=256$, $r=13$, $m=64$ を用いると,

$$T_S/T^Q = 30.3/v \tag{B.13}$$

$$T_C/T^Q = 0.75 \tag{B.14}$$

$$T_{com} = 834 \tag{B.15}$$

付 録 C

第 4 章の付録

C.1 QS 方式の処理量

Quisquater 等によって提案された依頼計算の性能解析は次のように行った. 512 ビットの法に対し, クライアントの乗算回数 N_0, サーバの乗算回数 N_1, 通信ブロック数 N_2は次の様に計算される.

$$N_0 = 3 \times 64 = 192 \tag{C.1}$$

$$N_1 = 8 \times 63 + 19 \times 64 = 1720 \tag{C.2}$$

$$N_2 = 2 + 12 \times 64 = 770 \tag{C.3}$$

N_0と N_2はプロトコルの仕様から自然に導ける. N_1の内容は, 63 回の 256 乗の計算（第 1 項）と, $B = \{0, 1, 2, 4, 5, 12, 20, 30, 33, 54, 77, 84, 230\}$を基底とする加算系列（Addition sequence）に基づいた 64 回のべき乗剰余計算（第 2 項）からなる. この加算系列を用ると, 256 乗までのべき乗は高々19 回の乗算で計算できる. 従って正規化された処理時間は

$$F = 192 + 1720/R_1 + 770/R_2 \tag{C.4}$$

表 C.1に種々の R_1, R_2に対する Fの値を示す.

表 C.1: QS 法の正規化処理時間

$R_1 \setminus R_2$	1	10	10^2	10^3	10^4	10^5
1	2682	1989	1920	1913	1912	1912
10	1134	441	372	365	364	364
10^2	979	286	217	210	209	209
10^3	964	271	201	194	194	194
10^4	962	269	200	193	192	192
10^5	962	269	200	193	192	192

付 録 D

第 5 章の付録

D.1 出力が運ぶ情報量

問い合わせベクトルの長さ i の系列が与えられたとき, 各ベクトルに対するクライアントの応答は OK または NG だから, 任意のクライアントのほ 2^i 個の葉 (leaves) を持った二進木 (Binary tree) で表せる. 一つのノードから伸びている 2 つの枝はそれぞれ OK, NG に対応する. i 個のクライアントの応答を i 次元の二値ベクトル $O_i = [o_1, o_2, \cdots, o_i]$ で表す. 各要素は OK, NG に対応して 0 または 1 の値をとるとする. ある履歴 O_i が観測される確率を $Q(O_i)$ で表す. また $(i-1)$ 回目までの履歴が O_{i-1} であるという条件の下で i 回目の応答が o_i である確率を

$$P(o_i|O_{i-1}) \tag{D.1}$$

で表す. 例えば $P(0|O_{i-1})$ はそれまでの履歴が O_{i-1} であるという条件の下で, クライアントが OK と応答する確率を表す. さらに, 与えられたベクトル O_i の最初の j 個の要素で新たに j 次元ベクトルを作る関数を σ_j で表す. 例えば, $\sigma_j(O_i) = [o_1, \cdots, o_j]$, 但し, $j \leq i$. 次の等式は任意の正整数 i について成り立つ. 従って,

$$\sum_{o_i=0}^{1} P(o_i|O_{i-1}) = 1 \tag{D.2}$$

$$\sum_{O_i} Q(O_i) = 1 \tag{D.3}$$

$$Q(O_i) = \prod_{j=1}^{i} P(o_j|\sigma_{j-1}(O_i)) \tag{D.4}$$

ここで, o_j は O_i の j 番目の要素とする.

次に, 特定の履歴 O_i が観測された時にまだ候補に残っている D の数を $N(O_i)$ で表すと, $N(O_i) = Q(O_i)N_0$ が成り立つ.

$$E[\log_2 N(O_i)]$$

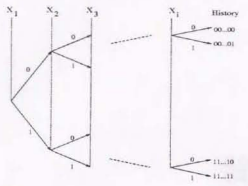


図 D.1: i-ラウンド攻撃の履歴

$$
\begin{aligned}
&= \sum_{O_i} Q(O_i)\log_2 Q(O_i)N_0 \\
&= \sum_{O_i} Q(\sigma_{i-1}(O_i))P(o_i|\sigma_{i-1}(O_i))\cdot\log_2 Q(\sigma_{i-1}(O_i))P(o_i|\sigma_{i-1}(O_i))N_0 \\
&= \sum_{O_{i-1}}\sum_{o_i=0}^{1} Q(O_{i-1})P(o_i|O_{i-1})\cdot\left\{\log_2 Q(O_{i-1})N_0+\log_2 P(o_i|O_{i-1})\right\} \\
&= \sum_{O_{i-1}} \log_2 Q(O_{i-1})N_0 + \\
&\quad \sum_{O_{i-1}}\left\{Q(O_{i-1})\sum_{o_i=0}^{1} P(o_i|O_{i-1})\log_2 P(o_i|O_{i-1})\right\} \\
&= E[\log_2 N(O_{i-1})] - \sum O_{i-1}Q(O_{i-1})h(P(0|O_{i-1})) \qquad \text{(D.5)}
\end{aligned}
$$

以上より,

$$
\begin{aligned}
I(D;O_i)-I(D;O_{i-1}) &= E[\log_2 N(O_{i-1})]-E[\log_2 N(O_i)] &\text{(D.6)}\\
&= E[h(P(0|O_{i-1}))]\le 1 &\text{(D.7)}
\end{aligned}
$$

よって, 一つの応答が運んでいる Dに関する情報量は平均すると高々1ビットである.

D.2 組分けアルゴリズム

攻撃手順のステップ 2 での処理量を評価するために, ステップ 2 で扱われるのと等価な問題を考察しよう. k_0個の要素からなる集合 Sを考える. 各要素には 1 から 8 までの中から選ばれた整数が割り付けられているものとする. つぎに, 我々にできる事は一度に 2 つの要素を取り出して, その二つが同じ値を持つか否かを比較することだけだとする. この時各要素の持つ値が何かはわからないものとする. このような前提の下で我々が知りたいのは, 集合 Sを各要素の持つ整数値に従って, S_1から S_8までの 8 つの部分集合に分けるために必要な比較の回数である.

次のような組分けのアルゴリズムを用いた時の比較の回数を数えることにする. ただし, アルゴリズムの中で使われている「未決の要素」という言葉は, まだどの部分集合に属するか決定されていないすべての要素を指す. 最初は k_0個全ての要素が未決の要素である.

アルゴリズム
for i=1 to 7 do:

1: 未決の要素の中から任意に一つの要素 x を選ぶ.

2: x を他の未決の要素全てと比較する.

3: x と同じ値を持つ要素の集合を S_iとする. (x は S_iに属す)

next i
まだ未決の要素を集め S_8とする. (終)

ステップ 2 での比較の回数 Kを評価するために, 部分集合 S_iの要素の個数を k_iとする.

$$K=(k_0-1)+\sum_{i=1}^{7}\left\{k_0-\left(\sum_{j=1}^{i-1}k_j\right)-1\right\} \qquad \text{(D.8)}$$

Kは k_jの値に依存するが, いま, $k_1=k_2=\cdots=k_8=k_0/8$, $k_0=\lceil 127/2\rceil=64$ と仮定すれば,

$$K=\frac{35}{8}k_0-7=273 \qquad \text{(D.9)}$$

本文中の攻撃では上記アルゴリズムは 2 回使われるので, 全体で約 546 回の比較が行われる.

D.3 攻撃対策の効果

まず付録 D.1と同様の 2 進木を考えよう. 但し, 今回は許容できる NG の回数の上限 u を設けているので 2 進木のいくつかの枝は刈り込まれた形となる. NG の観測確率は一定で qとする. プロトコルが停止する応答 (u 回目の NG に対応するノード) を停止ノードと呼ぶことにする. ルートノードから停止ノードまでのパスは丁度 u 回の NG 応答を含んでおり, しかも最後のクライアントからの応答は NG である. 攻撃者は t 回に亘ってプロトコルから逸脱し, 秘密情報を取りだそうとするものとする. このとき, t 回目の問い合わせが終わった段階でまだプロトコルが停止していないノードを生き残りノードと呼ぶ. i 回目の問い合わせでプロトコルが停止する確率を P_iで表す. P_iは式 (5.17) で表される負の二項分布に従う. 次に i 回目の問い合わせで停止ノードとなった場合にそれまでに得られる情報量を h_iで表す. また t 回の問い合わせで生き残りノードが観測された場合に得られる情報

量を h_i で表す. 生き残りノードの観測される確率は

$$1-\sum_{i=0}^{t} P_i \tag{D.10}$$

であるから, 結局次式が成り立つ.

$$I(D;O)=\sum_{i=0}^{t} h_i\cdot P_i + h_t\cdot\left(1-\sum_{i=0}^{t} P_i\right) \tag{D.11}$$

もし, $u/q \ll t$ が成り立つならば, 第二項は無視でき, t は無限大とみなして良い. 従って,

$$I(D;O)\simeq\sum_{i=0}^{\infty} i\cdot P_i=\frac{u}{q} \tag{D.12}$$

さらに, 標準偏差σ_Iは次式で与えられる.

$$\sigma_I=\frac{p}{q}\sqrt{u} \tag{D.13}$$

付 録 E

第6章の付録

E.1 クラス1攻撃の零知識性

クライアントは A を Z_L^* から一様ランダムに選んでいる. これに対しシミュレータは奇数$\tilde{A}$を $Z_{N'}$ から一様ランダムに選ぶ. 但し, $N'=(N-1)/2$. $|\{A\}|$ および $|\{\tilde{A}\}|$ をそれぞれ n_0, n_1で表す. $\{A\}\subset\{\tilde{A}\}$ だから $n_0<n_1$である.

$$\begin{aligned} n_0 &= L\left(1-\frac{1}{2}\right)\left(1-\frac{1}{P'}\right)\left(1-\frac{1}{Q'}\right) \\ &= (P'-1)(Q'-1) \end{aligned} \tag{E.1}$$

$$n_1 = \frac{N'}{2}=\frac{2P'Q'+P'+Q'}{2} \tag{E.2}$$

$m=\log_2 N$とする. 任意のサーバS'に対してプロトコルが系列αを出力する確率を入力サイズmをインデックスとして$\pi_{0,m}$で表し, また, シミュレータが系列αを出力する確率を$\pi_{1,m}$で表す. 分布$\pi_{0,m}$に従って選ばれた文字列αに対して, 任意の多項式時間検査マシンTが1を出力する確率$p_0^T(m)$で表す. シミュレータに関しても確率を$p_1^T(m)$を同様に定義する.

$$\begin{aligned} \pi_{0,m}(\alpha) &= \sum_{A} Prob(A)\cdot\pi_{0,m}(\alpha|A) \\ &= \sum_{A}\frac{1}{n_0}\cdot\pi_{0,m}(\alpha|A) \end{aligned} \tag{E.3}$$

$$\begin{aligned} \pi_{1,m}(\alpha) &= \sum_{\tilde{A}} Prob(\tilde{A})\cdot\pi_{1,m}(\alpha|\tilde{A}) \\ &= \sum_{\tilde{A}}\frac{1}{n_1}\cdot\pi_{1,m}(\alpha|\tilde{A}) \end{aligned} \tag{E.4}$$

本文中で示したように, $A,\tilde{A}$に依存しない共通の確率分布$\pi(\alpha^-)$を用いて, $\pi_{0,m},\pi_{1,m}$は次のように書ける.

$$\pi_{0,m}(\alpha|A)=\pi(\alpha^-)\pi'_{0,m}(O|\alpha^-,A) \tag{E.5}$$

$$\pi_{1,m}(\alpha|\tilde{A}) = \pi(\alpha^-)\pi'_{1,m}(\mathcal{O}|\alpha^-, \tilde{A}) \tag{E.6}$$

シミュレータの仕様から A と等しい $\tilde{A}$ に対しては,

$$\pi'_{0,m}(\mathcal{O}|\alpha^-, A) = \pi'_{1,m}(\mathcal{O}|\alpha^-, \tilde{A}) \tag{E.7}$$

が成り立つ. よってこの時

$$\pi_{0,m}(\alpha|A) = \pi_{1,m}(\alpha|\tilde{A}) \tag{E.8}$$

従って,

$$\begin{aligned}
\Delta &\stackrel{\text{def}}{=} \left|p_0^T(m) - p_1^T(m)\right| \\
&= \left|\sum_\alpha \pi_{0,m}(\alpha) Prob\left(T(m,\alpha)=1\right) - \sum_\alpha \pi_{1,m}(\alpha) Prob\left(T(m,\alpha)=1\right)\right| \\
&= \left|\sum_\alpha \left\{ \sum_{\tilde{A}\in\{A\}} \left(\frac{1}{n_0} - \frac{1}{n_1}\right) \pi_{0,m}(\alpha|A) Prob\left(T(m,\alpha)=1\right) \right.\right. \\
&\qquad \left.\left. - \sum_{\tilde{A}\notin\{A\}} \frac{1}{n_1} \cdot \pi_{1,m}(\alpha|\tilde{A}) Prob\left(T(m,\alpha)=1\right) \right\}\right| \\
&= \left|\left(\frac{1}{n_0} - \frac{1}{n_1}\right) \sum_{\tilde{A}\in\{A\}} \sum_\alpha \pi_{0,m}(\alpha|A) Prob\left(T(m,\alpha)=1\right) \right. \\
&\qquad \left. - \frac{1}{n_1} \sum_{\tilde{A}\notin\{A\}} \sum_\alpha \pi_{1,m}(\alpha|\tilde{A}) Prob\left(T(m,\alpha)=1\right)\right| \\
&\le \left(\frac{1}{n_0} - \frac{1}{n_1}\right) \cdot n_0 + \frac{1}{n_1} \cdot (n_1 - n_0) \\
&= 2 \cdot \frac{n_1 - n_0}{n_1} = \frac{6(P'+Q') - 4}{2P'Q' + P' + Q'} = O(2^{-\frac{n}{2}})
\end{aligned} \tag{E.9}$$

従って, 全ての定数 c と充分に大きい m に対して, $\Delta < 1/m^c$であり, シミュレータの系列はプロトコルの系列と多項式時間識別不能である. これはクラス 1 攻撃によってなんら新たな知識が漏れることがないことを意味する.

E.2 クラス 2 攻撃に対するシミュレータ

[シミュレータ M_2]

(1) M, Nを S'に与える.

(2) $\boldsymbol{X}_1$を S'から得る.

(3) $(b-1)$ 個の乱数 $R_{1,j}, (j=1,\cdots,b-1)$ を Z_N^*から生成し,

$$\begin{cases} Y_{1,1} &= R_{1,1} \cdot M \bmod N \\ Y_{1,j} &= R_{1,j} \bmod N \quad (j=2,\cdots,b-1) \end{cases} \tag{E.10}$$

で $\boldsymbol{Y}_1$を定義し, S'に与える.

(4) S'から $\boldsymbol{X}_2$を得る.

(5) $Z_{N'}$ (但し, $N' = (N-1)/2$) から$\widetilde{B}$を一様ランダムに生成 ($\widetilde{B} = \sum_{i=0}^{l-1} \tilde{b}_i \cdot b^i$). 次の様に $\boldsymbol{Y}_2$を計算し, S'に与える.

$$Y_{2,j} = R_{2,j} \cdot \prod_{i:b_i=j} \widetilde{X}_{2,i} \bmod N \tag{E.11}$$

(6) S'から Zを得る.

(7) $\widetilde{T} = Z \cdot R^{-1}$を計算. 但し, R^{-1}は正しいプロトコルの場合の定義と同じ.

(8) $T = M^{\widetilde{B}} \bmod N$ を計算.

(9) $\tau = \widetilde{T}/T \bmod N$として, 次の規則で$\mathcal{O}$を決める.

$$\mathcal{O} = \begin{cases} OK & (\tau = \pm 1) \\ NG & (\tau \neq \pm 1) \end{cases} \tag{E.12}$$

(10) $\alpha = (N, M, \boldsymbol{X}_1, \boldsymbol{Y}_1, \boldsymbol{X}_2, \boldsymbol{Y}_2, Z, \mathcal{O})$ を出力. (シミュレータ終)

E.3 補題 1 の証明

ステップ 8 の検算手順によって次の法則が成り立つ:

$$\begin{aligned} (\epsilon = \pm 1) &\rightarrow \mathcal{O} = OK \\ (\epsilon \neq \pm 1) &\rightarrow \mathcal{O} = NG \end{aligned}$$

α, βに関する次の 3 つの命題は真である.

$$\begin{aligned} \alpha \in G_0 \cap \beta \in G_0 &\rightarrow \epsilon = \pm 1, \text{従って} \mathcal{O} = OK \\ \alpha \in G_0 \cap \beta \notin G_0 &\rightarrow \epsilon = \pm\beta \neq \pm 1, \mathcal{O} = NG \\ \alpha \notin G_0 \cap \beta \in G_0 &\rightarrow \epsilon = \pm\alpha^B \neq \pm 1, \mathcal{O} = NG \end{aligned}$$

最後の命題については $B \in Z_L^*$であることに注意すれば, Bに対して $B \cdot C \bmod L = 1$ なる Cが存在し, $\pm\alpha^B = \pm 1$ と仮定して両辺を C乗すると$\pm\alpha = \pm 1$ となり命題の仮定 ($\alpha \notin G_0$) に反することで証明できる. (証明終)