強制的分散コンピューティングによる暗号解読とその有効性

森藤賢司* 森口一郎**

アルバート＝ラズロ・バラバシらにより提案された強制的分散コンピューティングの手法を改良し、より効率的な計算を可能とする強制的分散コンピューティングの手法を提案する。次に、改良した手法で、部分和問題を応用した公開鍵暗号方式であるMerkle-Hellmanナップサック暗号により作成された暗号文を解読させて、その実用性を検証した。

キーワード：寄生コンピューティング，TCP/IP，チェックサム，部分和問題，Merkle-Hellman ナップサック暗号，分散コンピューティング

Deciphering by Forced Grid Computing and the Effectiveness

Kenji MORITO and Ichirou MORIGUCHI

The technique of Forced Grid Computing proposed by Albert-László Barabási et al. is improved. And a new technique of Forced Grid Computing is proposed, which enables more efficient calculation. Secondly, the practicality of the improved technique is verified by applied to deciphering the cryptogram encoded by Merkle-Hellman Knapsack Cryptosystem which is one of the public-key cryptography that uses the subset sum problem.

Keyword：Parasitic Computing, TCP/IP, Checksum, Subset Sum Problem, Merkle-Hellman Knapsack Cryptosystem, Grid Computing

---

*東京情報大学総合情報学部情報システム学科
Tokyo University of Information Sciences, Faculty of Informatics, Department of Information Systems
**東京情報大学総合情報学部情報システム学科
Tokyo University of Information Sciences, Faculty of Informatics, Department of Information Systems

2007年6月29日受理

1．はじめに

現在、1台のコンピュータでは計算不可能な大規模計算を実現するために、SETI@home（http://www.seti.org/）などの、さまざまな分散コンピューティングプロジェクトがインターネット上で行われている。これらの分散コンピューティング手法では、サーバから与えられた問題を解くために、対応したクライアントソフトのインストールが必要不可欠である。逆に考えると、クライアントソフトをインストールしなければ、その分散コンピューティングプロジェクトに参加できないということでもある。そのなかで、ノートルダム大学のアルバート＝ラズロ・バラバシらが2001年に提案したParasitic Computing[1]手法（以下、「強制的分散コンピューティング」と呼ぶ）は、クライアントを必要とせず、ネットワーク上のあらゆるコンピュータを利用して強制的に計算させることが可能であるため、通常の分散コンピューティング手法のサーバとクライアントの関係を根底から覆した。

強制的分散コンピューティングでは、ネットワーク内で通信されたデータの整合性を保障する手法であるチェックサムを利用する。チェックサムとは、受信したパケットを加算演算することにより、整合性を判断することができる簡単な手法である。強制的分散コンピューティングでは、この加算演算を利用し、考えられる解の組合せすべてを送信することにより、相手のノードにそれが正しい解か否かを判定させることで計算を可能としている。もしそれが正しい解であれば、相手のノードはそのパケットを上位レイヤに渡し、そのレイヤのプロトコルに対応したレスポンスを返信する。正しくない解であれば、相手ノードはそれを破棄する。つまり、計算を強制されたノードは、ただプロトコルどおりの動作をしているだけである[1]。

強制的分散コンピューティング手法におけるクライアントとは、巨大なネットワークであるインターネット上に公開された無数のコンピュータである。もし、この手法が実現するのであれば、分散処理に参加するためにクライアントをインストールするなどの、わずらわしい作業が不要となる。しかし、この手法自体には効率という面での大きな疑問があり、それらを検証できる定量的データやプログラムなども存在しない。

本論文では、TCP/IP（Transmission Control Protocol/Internet Protocol）を利用し、より効率的な計算を可能とする強制的分散コンピューティングの手法を提案する。次に、部分和問題を応用した公開鍵暗号方式である、Merkle-Hellmanナップサック暗号[6][7]により作成した暗号文を、秘密鍵を使用せず公開鍵のみで解読させ、その実行結果よりこの計算モデルの有効性について考察する。

本論文では強制的分散コンピューティングの技術上の問題のみに焦点を当て、この手法の倫理的、法律的問題は議論しない。

2．強制的分散コンピューティング

本章では、参考文献[1][2]により提案された強制的分散コンピューティングの手法を説明し、次に、それを改良した、より効率的な手法を説明する。

2.1　バラバシ強制的分散コンピューティング

(1) 解かせるべき問題を持っているノード（Parasite node）は、あらかじめ複数のHTTP（Hyper-Text Transmission Protocol）サーバである計算をさせたいノード（Target node）とのTCP/IPコネクション確立しておく。図1は、計算をさせたいノードを4つ用意し、これらのノードで問題を計算する例である。
(2) 問題を持っているノードは、問題の解となりうる組合せを格納したパケットを図1の(1)のように、計算をさせたい各ノードの80番ポートあてに送出する。そのパケットを受信し

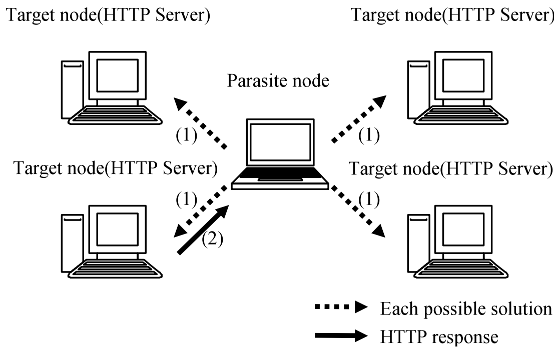


図1　強制的分散コンピューティング　(1)指定した複数のターゲットに、チェックサムを改変したパケットを送信する。(2)正しいチェックサムである場合、その送信したパケットが問題の解である。

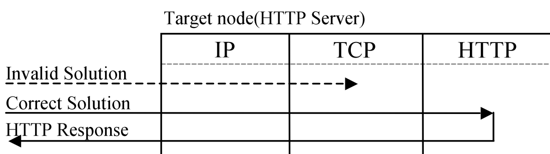


図2　組合せパケット受信したノード　正しい解でない場合は、TCPチェックサムにより、整合性がとれず破棄される。正しい解である場合は、HTTPサーバに渡され、HTTPサーバはレスポンスを返す。

たノードは、プロトコルで定められた処理を行う。正しい解でないパケットはTCPレイヤのチェックサムにより、整合性がとれず破棄される。正しい解であるパケットは、TCPレイヤのチェックサムにより、整合性を確認し、上のレイヤにあるHTTPサーバにTCPのデータ部を渡す。HTTPサーバでは、データ部の内容に従ったレスポンスを返す（図2）。

(3) 送信したノードから図1の(2)のように、何らかのHTTPレスポンスが返ってきたなら、それがその問題の解である。

このように、個々の組合せを他のノードに送信し、チェックサム計算での判定結果により充足可能性問題や部分和問題のような組合せ探索問題を解くことができる。

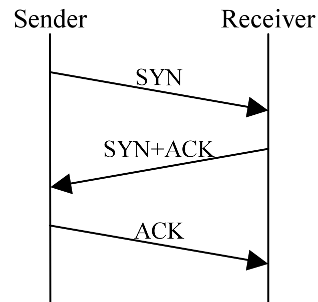


図3　TCP 3ウェイ・ハンドシェイク　送信元ノードはあて先ノードへSYNパケットを送出する。あて先ノードはSYNパケットで指定されたあて先ポート番号が待ち状態であればSYN+ACKを送信元ノードへ送信する。最後に送信元ノードがACKを送信することでコネクションが確立される。

しかし、この手法ではコネクションの確立が必要であることや、順序識別方法がなく、各ノードに、一度に複数のパケットを処理させることができないなど、あまり効率的ではない。

2.2　効率性を重視した手法の提案

TCP（Transmission Control Protocol）では、通信開始時に、図3のような3ウェイ・ハンドシェイク（3-way handshake）と呼ばれるコネクション確立を行うが、本研究での改良では、このコネクションそのものを利用することにした。

解きたい問題を持っているノードは、コネクションを開始時に送信するSYNパケットに問題の解になると考えられるすべての組合せをひとつずつ格納し、強制的に計算させるノードへ送信し続けるようにする。そして、問題の解であるSYN+ACKパケットが、何番目にあて先ノードへ送信したパケットか識別できるように、送信元ポート番号（Source port number）を使用して識別する。送信元ポート番号を使用する理由は、あて先ノードのあて先ポート番号（Destination port number）に何もサーバソフトのソケットがバインドされていない場合、あて先ノードからはRSTパケットが返信されるからである[3]。このRSTパケットには本来、

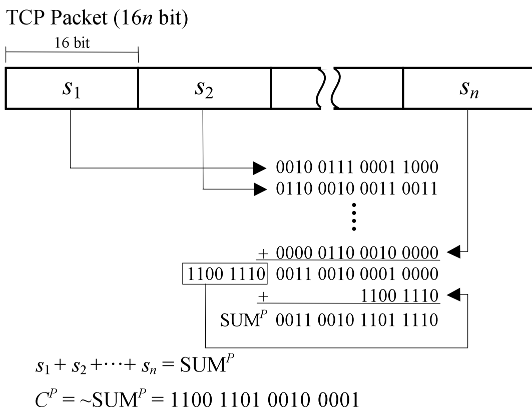


図4　TCPチェックサムの計算

TCPにおいて順序識別に使用される順序番号（Sequence number）が0の状態で送信されてくる。このような場合に対処するために送信元ポート番号を順序番号として使用することにした。これにより、あて先ノードが何らかのサーバでない場合でも強制的に計算させることが可能となる。

ここで、送信元ポート番号として0～1023のWell-knownポート番号を使用するとパケットが返ってこない場合があるため使用しない。改良手法での送信元ポート番号はマスクが容易であることや安全性を考慮し、予約なしポート番号である49152～65535を使用する。

3．改良手法による探索問題解決へのアプローチ

本章では、改良した強制的分散コンピューティングにより組合せ探索問題を解決する手法を説明する。

3.1　TCP/IPチェックサム

強制的に計算させるためのパケットを作成するためには、TCPでのチェックサムの計算を理解する必要がある。以下はTCP/IPにおいてのチェックサムの計算の説明である。

(1) 送信側での処理

TCPヘッダのチェックサムフィールドには0を格納しておく。次に、擬似ヘッダ（Pseudo-header）、TCPヘッダ、TCPデータを、16bit区切りの値として、それらを１の補数加算した値の１の補数を、16ビットであるTCPヘッダのチェックサムフィールドに代入する[1][3][4]。

(2) 受信側での処理

擬似ヘッダ、TCPヘッダ、TCPデータを、16bit区切りの値として1の補数加算し、そのビットが「$FFFF_{(16)}$」となればそのTCPパケットは正しいものとして上位レイヤに届けられる。その値がそれ以外のものであれば、破棄する[1][3][4]。

図4はチェックサムの計算例である。まずTCPパケットを16bitずつに区切り$s_1, s_2, \cdots, s_n$とする。次に、それらの値を１の補数加算した値SUM^Pを求める。その１の補数の値であるC^PをTCPヘッダのチェックサムフィールドに代入する。１の補数加算とは桁あふれしたビットを下位ビットに図のように加算する方法である。

3.2　送信時チェックサム計算の簡略化

強制的に計算させたいノードへパケットを送信するたびに**3.1**の(1)の処理を繰り返すのは非効率的である。したがって、この処理を一度だけに済ませる方法を説明する。なお、チェックサム計算に必要な、擬似ヘッダと送信元ポート番号、チェックサムフィールド以外のTCPヘッダの構築は既に完了しているものとする。

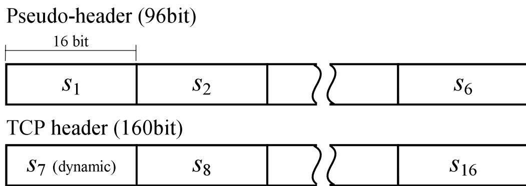
(1) 静的なヘッダフィールドのチェックサム

まず、**3.1**(1)の方法により、擬似ヘッダ、TCPヘッダのみのチェックサムC^Hをもとめる（図5(1)）。このとき、チェックサムフィールドと動的であるヘッダフィールドはまだ格納される値が確定しないため、0を格納しておく。次に、もとめたC^Hをデータ部に格納しておく。

(2) 動的なヘッダフィールドのチェックサム

送信するたびに、値が変化するヘッダフィールドに関しては、データ部にその補数の値を格納する。

これらの方法により、組合せを格納する前ま



$$s_1 + s_2 + \cdots s_6 + 0 + s_8 + \cdots + s_{16} = \sim\mathrm{SUM}^H = C^H \quad (1)$$

$$s_1 + s_2 + \cdots + s_{16} + C^H + \sim s_7 = \mathrm{FFFF}_{(16)} \quad (2)$$

図5　ヘッダ部チェックサムの無効化　ヘッダ部のみのチェックサムC^Hや動的なヘッダフィールドの補数をあらかじめ求めることで、送信時のチェックサム計算を不要にする。

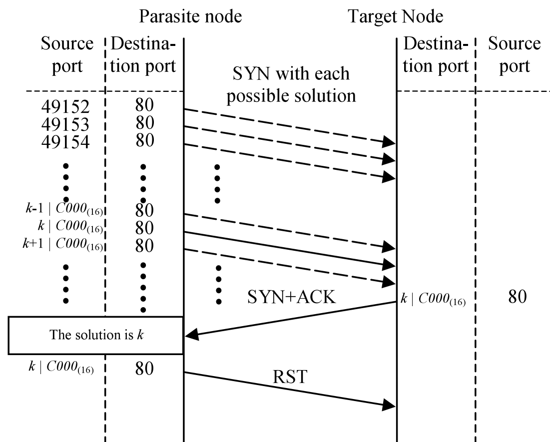


図6　3ウェイ・ハンドシェイクを利用した計算
ポート番号を利用しパケットを識別する。

での1の補数和を必ず$\mathrm{FFFF}_{(16)}$にすることができる（図5(2)）。したがって、送信する際に毎回チェックサムを計算する必要がなくなる。

3.3　手順

3.3.1　強制判定SYNパケットの構築と送信

(1) 事前準備

あらかじめ、擬似ヘッダと送信元ポート番号、チェックサムフィールド以外のTCPヘッダを構築し、**3.2**(1)の方法で静的なヘッダフィールドのチェックサムをもとめ、期待する1の補数和をチェックサムフィールドに格納しておく。

(2) 組合せパケットの構築

順序識別に使用する送信元ポート番号は、順序番号jと$\mathrm{C000}_{(16)}$とのビットごとの論理和 j | $\mathrm{C000}_{(16)}$ で算出する。これによりポート番号は、49152〜65535の範囲内となる。**3.2**(2)の方法で送信元ポート番号の補数を格納する。これらの構築が終れば順次組合せを格納する。

(3) 送信

通常のソケットを使用した場合、チェックサムを自由に改変することが不可能である。そのためにTCP以下のレイヤであるIPパケットやEthernetフレームを任意に作成できるソケットであるRaw Socketを用いて(2)で作成したTCPパケットを送信し続ける。

3.3.2　SYN+ACK、RSTパケットの受信

送信を開始してしばらくすると、あて先ノードのうち問題が解けたノードからSYN+ACKもしくはRSTパケットが返信される。このパケットに格納されているあて先ポート番号dと$\mathrm{C000}_{(16)}$とのビットごとの排他的論理和$d \oplus \mathrm{C000}_{(16)}$により、順序番号jが算出できる。問題が解けると、これ以上組合せを送信する必要がないのでSYNの送信を停止する。

図6はk番目に送ったパケットが正しい解であったときの例である。解かせたい問題を持っているノードは送信元ポート番号を49152, 49153, …と変更し、パケットを送信し続ける。k番目に送ったパケットが送信先ノードのTCPチェックサムにより整合性が確認され、送信先ノードは3ウェイ・ハンドシェイクを行おうとSYN+ACKをあて先ポート番号 k | $\mathrm{C000}_{(16)}$ へ送信する。そのあて先ポート番号と$\mathrm{C000}_{(16)}$との排他的論理和により、k番目に送ったパケットであると判断できる。

3.4　分散処理

これらの処理は単一ノードでも可能であるが、短時間で大量のパケットをひとつのノードに送ると特定のターゲットネットワークの

SYNトラフィックが増大し、IDS（Intrusion Detection System：侵入検知システム）や管理者にアタックであると認識されてしまう可能性がある。またQoS（Quality of Service）でSYNパケット総量に制限がかけられていることもある。そこで、組合せの送信を複数のノードに分散することにした。各ノードへ送信するパケットの送出間隔は、Linux上のusleep関数で停止可能な最短時間である約10msとする。

4．Merkle-Hellmanナップサック暗号の解読

本章では、TCP/IP上で公開鍵と暗号文を用いて全探索により暗号を解読する方法を説明する。ここで必要になる公開鍵と、それにより暗号化された暗号文の作成のみを説明する。

4.1 公開鍵の作成

参考文献[5][6]より、暗号化に必要な公開鍵の作成について説明する。なお、鍵サイズはnビットとする。

手順1）

$2 \leq j \leq n$について、

$$a'_j > \sum_{i=1}^{j-1} a'_i \tag{1}$$

を満たす、超増加である乱数の列a'_1～a'_nを作成し、秘密のベクトル$\vec{a}'_1=(a'_1, \cdots, a'_n)$を作成する。

次に、互いに素である大きな乱数wとmを作成する。mは、

$$m \geq \sum_{i=1}^{n} a'_i \tag{2}$$

を満たす必要がある。

手順2）

手順1で作成した秘密のベクトル$\vec{a}'$、wとmから

$$a_i = a'_i \cdot w \pmod{m} \tag{3}$$

により、公開鍵のベクトル$\vec{a}=(a_1, \cdots, a_n)$を作成する。

ところで、TCPにおいて、チェックサムは**3.1**で示したように、16ビット区切りで計算され、16ビットの整数として表現される。すなわち、すべてを加算した結果が2^{16}以上だったときに、桁あふれにより、まったく異なるパケットでも、同じチェックサムになる場合がある。この桁あふれを防ぐために、

$$\sum_{i=1}^{n} a_i < 2^{16} \tag{4}$$

を満たす公開鍵を作成しなければならない。

これらの条件(1)(2)(4)を満たす、最大の鍵サイズnは12である。

4.2 暗号文の作成

参考文献[5][6]より、暗号化について説明する。なお、鍵サイズはnビットとする。

手順1）

暗号化したい２進数のデータを、n桁の2進数に区切り、それらを0と1のベクトル$\vec{x}_1=(x_{11}, \cdots, x_{1n})$, $\vec{x}_2=(x_{21}, \cdots, x_{2n})$,…と考える。

手順2）

手順1で作成した、ベクトル$\vec{x}_1, \vec{x}_2$,…と、**4.1**の手順2で作成した、公開鍵のベクトル$\vec{a}$の内積$C_1=\vec{x}_1 \cdot \vec{a}$, $C_2=\vec{x}_2 \cdot \vec{a}$,…をもとめ、これらを暗号文とする。

以上により、C_1, C_2,…は公開鍵のベクトル$\vec{a}$の数のセット$a_1, \cdots, a_n$の部分和となる。したがっ

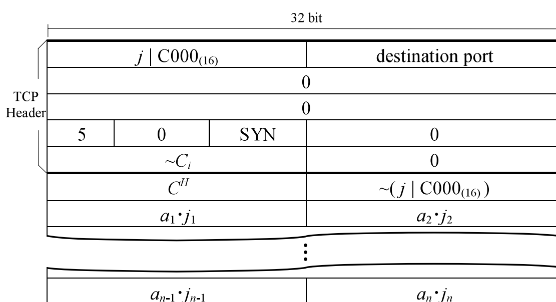


図7 送信するTCPパケットの構成

て、これらの部分和問題を解決するためには最大2^n回の試行が必要である。

4.3　強制的分散コンピューティングへの適用

3.3の手法を利用し、組合せ探索問題である部分和問題解決のためのTCPチェックサムへの適用と、その送信方法について説明する。

手順1)

3.3.1の(1)で示した方法によりパケットを構築する。部分和となっている暗号文C_iをTCPヘッダのチェックサムフィールドに格納する。順序番号jは0とする。

手順2)

3.3.1の(2)で示した方法により、送信元ポート番号を算出する。次に、順序番号jをn桁の0と1の数のセット$j_1, \cdots, j_n$とし、公開鍵の数のセット$a_1, \cdots, a_n$との積をとり、図7のようなTCPパケットを作成し、計算させたいノードに送信する。jに１を加算し、$j < 2^n$であれば、解であるSYN+ACKパケットを受信するまで手順2を繰り返す。

手順3)

受信した解をバッファにためておく。問題がまだある場合はiに１を加算し、手順2へ戻る。すべての問題が解決できたなら、バッファから解読した平文を出力し終了する。

5．実験と考察

本章では、本論文で提案した手法の有効性を確認するために、実際のネットワーク上で計算させ、その計算にかかる時間により強制的分散コンピューティングの有効性について考察する。

5.1　実験環境

まず実験に使用したコンピュータや、ネットワークの環境について説明し、次に実験に使用したプログラムを解説する。

表1表2に問題送信に使用したコンピュータの環境を示す。主な実験は表1のコンピュータを使用し、コンピュータの性能の差があたえる影響を検証するために表2のコンピュータを使用する。

この実験で使用するネットワークについては、倫理的正当性が確立していないので、インターネットを使用するのではなく、東京情報大学内ネットワークでのみ計算することにした。

表1　問題を送信するコンピュータの環境①

Model	IBM Intellistation EPro 6836-35J
OS	Vine Linux 3.2 Kernel-2.4.31
CPU	Intel Pentium III 933MHz
MEM	256MB PC133 SDRAM
Net IF	Intel 82559（Onboard）100Base-TX

表2　問題を送信するコンピュータの環境②

Model	EPSON AT-205
OS	Vine Linux 3.2 Kernel-2.4.31
CPU	Intel Celeron D 346（3.06GHz）
MEM	512MB PC3200 DDR SDRAM
Net IF	Intel 82562EZ（Onboard）100Base-TX

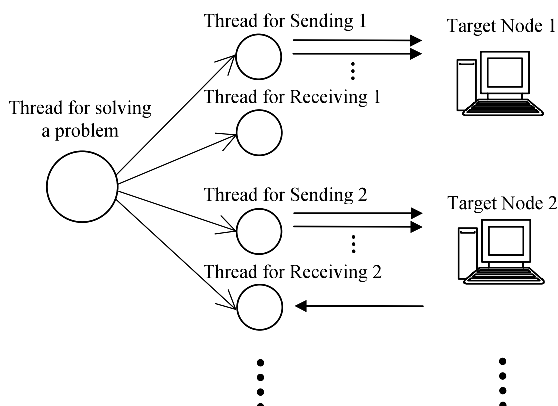


図8　送受信のスレッド処理　１つの部分和問題を解決するたびに問題解決用スレッド（Thread for solving a problem）を作成する。そのスレッドは指定されたノードに対応する送信用スレッド（Thread for Sending）と受信用スレッド（Thread for Receiving）を作成し送信、受信待ちを開始する。パケットを受信した場合、作成したすべてのスレッドを終了する。

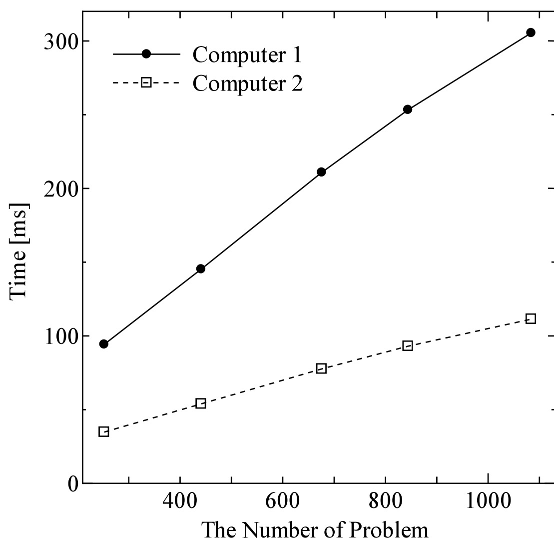


図9　CPUを使用した問題数に対する暗号解読時間

実験に使用したプログラムについての説明をする。このプログラムでは**4.3**の手法で、改良した強制的分散コンピューティングによる暗号の解読を行う。時間の計測については、プログラム開始直後から、終了直前までをこのプログラム内で測ることにする。図8は、本プログラムで問題を解決するためのパケットを送受信する際のスレッド処理の説明である。プログラムは、問題を解決するためのスレッドを作成する。そのスレッドは与えられたノードに対応する受信用スレッドと送信用スレッドを作成し、受信用スレッドで受信処理を開始したのち、送信用スレッドでパケットの送信を開始する。受信用スレッドにて対応したノードからのパケットを受信すると、その解をバッファに書き込み、問題解決のために作成したすべてのスレッドを終了する。なお、スレッド処理にはPOSIX Threadライブラリを使用した。

5.2　実験結果

まず、CPUを使用した一般的なコンピューティング手法により、総当り法で12bitの暗号文解読にかかった時間を計測した。

この一般的な手法による、暗号化により生じた部分和問題の数とその解決にかかった演算時間（ミリ秒）を図 9 に示す。図 9 より、表2のコンピュータ（Computer 2）は表1のコンピュータ（Computer 1）の約3倍の演算能力であることがわかる。

次に、強制的分散コンピューティングによる12bitの暗号解読にかかった時間を計測した。表1のコンピュータを使用した強制的分散コンピューティングによる、各部分和問題数別の送信ノード数と暗号解読にかかった時間（秒）を図10に示す。各ノードへの送出間隔は約10msであり、その送信を分散させたため、解読時間がノード数に反比例し、計算させるノード数を増加させることにより、計算時間が減少することがわかる。

次に、強制的分散コンピューティングにおける演算時間の限界を計測するために、送出間隔をとらず送信した。

表1のコンピュータを使用した送出間隔なしの強制的分散コンピューティングによる各部分和問題数別の送信ノード数と暗号解読にかかった時間（ミリ秒）を図11に示す。図11より、送出間隔なしの場合、送信ノード数による計算時間に図10のような明確な減少傾向が見られず、ほぼ横ばいであるため、表1のコンピュータ上での強制的分散コンピューティングにおける計算時間の限界であることがわかる。

最後に、コンピュータの性能差による演算時間の限界の影響を検証するために表2のコンピュータを使用した、送出間隔なしの強制的分散コンピューティングを行った。

表1のコンピュータ（Computer 1）と表2のコンピュータ（Computer 2）により、1台のノードを使用し、送出間隔なしの強制的分散コンピューティングで解読させた、問題数とその解決にかかった演算時間（ミリ秒）を図12に示す。図12と図9と比較すると、強制的分散コンピューティングではCPUの演算速度はそれほど影響しないことがわかる。

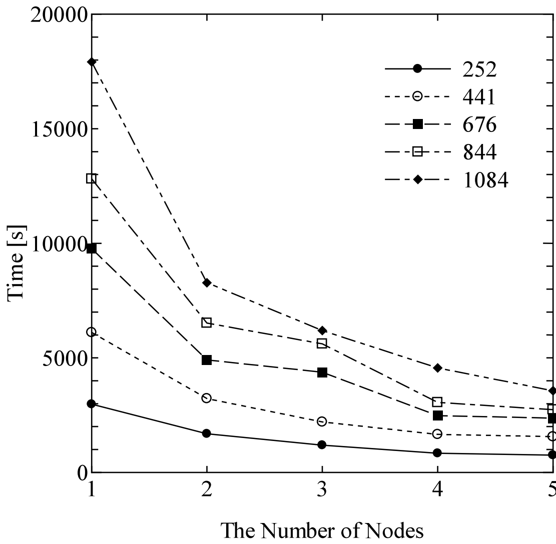


図10　分散数に対する暗号解読時間

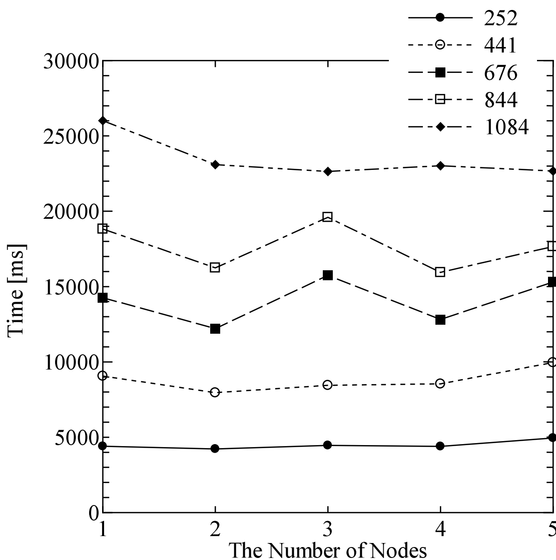


図11　分散数に対する暗号解読時間（送出間隔なし）

5.3　考察

この実験により、強制的分散コンピューティングによってMerkle-Hellmanナップサック暗号の解読、つまり部分和問題の解決が可能であることがわかった。しかし、16bit単位で計算をするチェックサムでは計算できる内容にかなり制限があることが問題である。より複雑な計算を行うためには、補助的にCPUによる計算を使わざるをえなくなるだろう。また、CPUによる一般的なコンピューティングと比較すると、計算時間が劣るということもわかった。

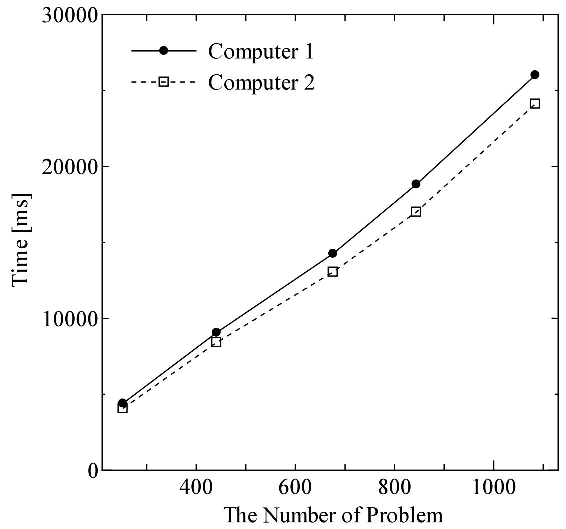


図12　改良手法による問題数に対する暗号解読時間

強制的分散コンピューティングによる計算時間に影響を及ぼす主な要因はCPUの演算速度ではなく、コンピュータのネットワークインタフェースやルータやスイッチャの単位時間当たりのパケット処理能力[pps]や単位時間当たりのビット伝送能力[bps]等の性能であると考えられる。

もしそうであるならば、ネットワークインタフェースの単位時間当たりのパケット処理能力と問題を解決するために作成されるイーサネットフレームの容量の積が、ネットワークインタフェースの単位時間当たりのパケット処理能力以下であると仮定し、ネットワークインタフェースの単位時間当たりのパケット処理能力をx、部分和問題を解決するために必要な試行回数をnとし、プログラムやOSのオーバヘッドをt_oとした場合の強制的分散コンピューティングによる計算時間t_Fは、

$$t_F = \frac{n}{x} + t_o \tag{5}$$

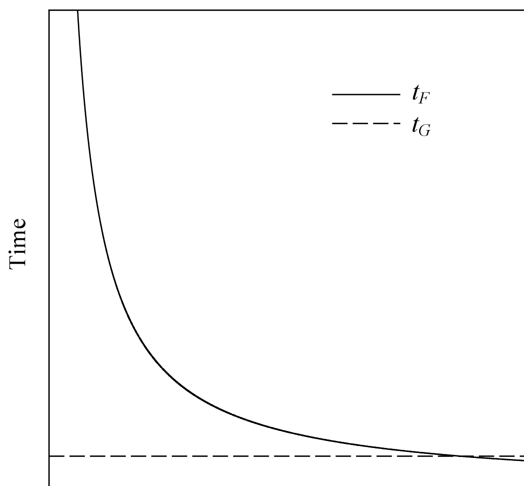


図13　部分和問題を解決する際の単位時間当たりのパケット処理能力に対する計算時間の予測
強制的分散コンピューティングによる計算時間をt_F、一般的なコンピューティングによる計算時間をt_Gとする。

と推測できる。

一般的なコンピューティングによる計算時間t_Gが$t_G > t_o$であるなら、図13に示すように一般的なコンピューティングによる計算時間を超える可能性があることになる。

6. おわりに

本論文では、バラバシらにより提案された強制的分散コンピューティングをより効率的な計算ができるように改良し、部分和問題を応用した公開鍵暗号方式であるMerkle-Hellmanナップサック暗号により作成した暗号文の解読を試みた。その結果、CPUを使用した一般的なコンピューティングによる計算時間を超えることはできなかった。だが、ネットワーク機器の今後の進化によっては、一般的なコンピューティングによる計算時間を超える可能性があるということがわかった。

参考文献

[1] Albert-László Barabási, Vincent W. Freeh, Hawoong Jeong and Jay B. Brockman：*Parasitic computing*, Vol.412, pp.894-897, Nature (2001)
[2] Albert-László Barabási, et al.：*Supplementary Material for Parasitic Computing*, http://www.nd.edu/~parasite/sup.pdf
[3] 波多浩昭：いまどきのソケットプログラミング, 日経BP社, pp.229-250 (2004)
[4] 笠野秀松、マルチメディア通信研究会偏：インターネットRFC事典, アスキー出版局, pp.519-557 (1998)
[5] Martin E. Hellman: *The Mathematics of Public-Key Cryptography*, Vol.241, No.2, pp.146-157, Scientific American (1979)
[6] Douglas R. Stinson, 櫻井幸一 監訳：暗号理論の基礎, 共立出版, pp.204-207 (1998)