圧縮器への秘匿情報埋め込み機能の実装とその評価

2001MT043 楓 基靖　　2001MT100 谷川 隆通
指導教員　真野 芳久

1　はじめに

近年あらゆるもののデジタルデータ化とインターネットの普及に伴いデジタルデータが一般に広く浸透してきている。また、マルチメディアデータ等の情報量の多大なデータに関しては、インターネットを通じて転送するにはコストが掛かり過ぎてしまう。その為、データを転送する際にはデータに圧縮をかけてより小さくすることで転送コストの軽減をすることが日常となってきており、世に存在する伝送データは圧縮が掛けられているものがほとんどである。

2　目的

データが多種多様に存在する中で、第三者に見られることなく秘匿情報を手軽で安全に相手に渡すことは困難であると言える。秘匿情報を別のあるデータに埋め込む事でデータを隠蔽する手法は、現在ステガノグラフィや電子透かしに代表されており、これらの方法に着目する。本研究では、秘匿情報の隠蔽を世に数多く出回っている圧縮データに適用することで、秘匿情報の付いたデータを浸透させることができると考え、これらの圧縮器に対して秘匿情報を埋め込む機能を実装し、その機能の調査をして行く。

3　埋め込み手法

圧縮時に秘匿情報を埋め込む際、埋め込まれた秘匿情報が容易に取り出せてしまう様な方法では秘密通信としての意味を成さない。例としてアーカイブ等に挙げられる単純に元データへ秘匿情報を付加する方法が存在する。また、埋め込みをする元データそのものに手を加えることは、致命的な情報欠損をもたらす可能性が高く意味が無い。(以下、圧縮元のデータを元データ、埋め込む秘匿情報をステゴデータ、埋め込み後圧縮データをステゴ圧縮データ、埋め込み機能を実装した圧縮器と復号器をそれぞれステゴ圧縮器、ステゴ復号器と呼ぶ。) ステゴ圧縮データを広く世に違和感なく浸透させるために、ステゴ圧縮データは一般的に世に出回っている復号器 (以下一般復号器) で完全に復号できなければならない。また、秘密通信先の相手だけにステゴデータが閲覧でき、それ以外の相手は閲覧できない事が必要である。具体的には**図 1** に示す関係を最低条件とし機能の実装をする。

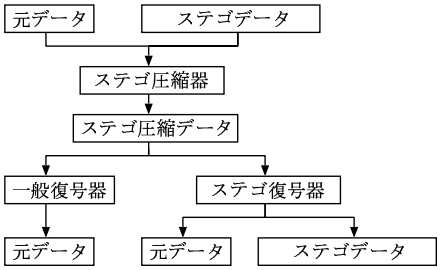

図 1　満たすべき最低条件

LHA 圧縮のアルゴリズムであるスライド辞書圧縮 (LZ77) 部分においてその圧縮時置き換え情報の冗長性 (相対位置と一致長) に着目して埋め込み機能を実現することは [3],[4],[5] で述べられているが、ここでは埋め込み機能を実装した圧縮器を一般に実用できる様にすることを重視し、これを実装する。(スライド辞書圧縮では、現在位置 (圧縮処理位置) からの文字列と現在までに出現した文字で構成された辞書部分にある文字列を比較し、最長一致した文字列を (辞書内位置情報, 一致文字列長) の対に置き換えることにより圧縮を行っている。) 本稿では [1] のソースコードを改変することによって埋め込み機能の実装をしていく。

3.1　最長一致文字列の選択による埋め込み方法

この節で取り扱う埋め込み方法は、最長一致文字列が複数存在した場合に生じる冗長性を利用したものである [3]。最長一致文字列選択による埋め込み方法の例を**図 2** に示す。

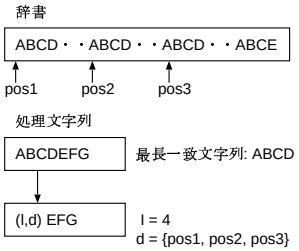

図 2　文字列の選択による埋め込み方法

pos1, pos2, pos3 からの 4 文字が最長一致文字列 ABCD となり、置き換える候補とできる。これらの候補の選択により、この場合では 1bit の情報を埋め込むことが可能である。例として、秘匿情報より得た bit 列が、0 の場合は pos1 を選択、1 の場合は pos2 を選択する (通常の圧縮では、最も処理位置から近い pos3 を選択している)。この方法で位置情報を選択し変更しても、圧縮時に位置情報と一致長に割り振られる bit 数は固定であるため、圧縮率の劣化は起こらない (LHA の lh5 圧縮法の場合、位置情報 13bit、一致長 8bit として圧縮情報が表現されている為、辞書内のどの位置情報を候補として取っても圧縮データとしては 13bit で固定と

なる)。ただし、実現する対象の圧縮器 LHA においては圧縮の第 2 段階として Huffman 符号化があるため、位置情報の選択次第では通常の圧縮と比べ圧縮率の劣化が起こる。

3.2 一致長操作による置き換え候補の拡大 1

ここでは圧縮情報である一致長に着目した埋め込み方法 [3] を述べる。3.1 節の方法では置き換える候補は全て最長一致文字列そのものであるが、辞書内にはより短い一致文字列が多数存在している可能性がある。最長一致長よりも短い部分文字列も置き換え候補とすることで埋め込み量の向上を図る。ただし、この方法を適用すると置き換える候補に最長一致よりも短い文字列を選択することになり、本来最長一致として置き換えられるべき文字が圧縮されないことになり、スライド辞書圧縮部分のみで圧縮率の劣化が起こる。

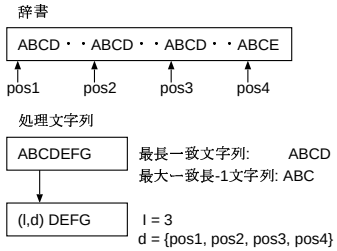

図 3　一致長操作による置き換え候補の拡大 1

図 3 は発見された最長一致文字列の一致長を (一致長 −1) として一致文字列を最長一致より短くした場合である。3.1 節の方法では pos4 は置き換えの候補から除外されるが、この方法により pos4 も一致文字列の候補の一つとして扱われ、埋め込み量が増えることになる。これは**図 2** と比べ埋め込められる情報が 1bit 増えているが、3.1 節の方法の置き換え情報 (4,d), d = {pos1, pos2, pos3} に対して この方法では (3,d)D, d = {pos1, pos2, pos3, pos4} となる為、文字 D 分だけ圧縮率の劣化が発生する可能性が高い。一般的に、一致長操作の度合に応じ圧縮率の劣化が顕著に現れると考えられる。

3.3 一致長操作による置き換え候補の拡大 2

3.2 節では、最終的に置き換えの候補となる文字列は一致長操作後の一致長の文字列だけであった (例として**図 3** の場合は一致長操作後の文字列は ABC) 。しかし、最長一致文字列も置き換え候補であることに変わりはない。同様に最長一致文字列から操作後の一致文字列までに存在する任意の長さの文字列でも置き換える事は可能である。最終的に操作されて得られた最も短い部分文字列から最長一致までの全ての長さの部分文字列を候補とすることで、置き換え候補を拡大し埋め込み量を増やす。

図 4 において、最長一致文字列は ABCD となり、1 だけ文字列を短くすると ABC が 3.2 節で述べられている置き換え候補の文字列であるが、ABCD(元の最長一致文字列) でもデータの置き換えは可能である。また、一致長を 2 だけ短くした場合も、置き換え候補文字列は一

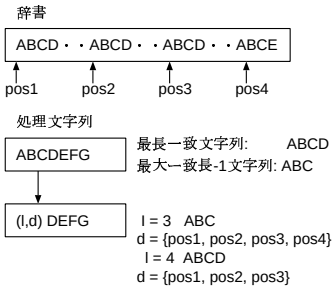

図 4　一致長操作による置き換え候補の拡大 2

致長操作後の文字列 AB と 元の最長一致文字列である ABCD の 2 つに加えて (最長一致長 −1 文字列) である ABC も候補として加わることになる。

3.4 木構造による未使用候補の有効利用

3.1 節から 3.3 節の方法では、置き換え候補の総数を n 個とすると、埋め込み量は $\lfloor \log_2 n \rfloor$bit となる。例えば、**図 2** の状況であると、埋め込める情報量は候補が 3 つであるため、 1bit である。仮に、候補の選択を pos1 と pos2 の 2 通りで 1bit を表すとしたならば、pos3 は候補であるにも関わらず、全く使用されないことになる。ここでは [5] による木構造を利用し、得られた候補全てに bit 列を割り振ることで埋め込み時に未使用となっている無駄な候補を有効に利用して埋め込み量を増やす。

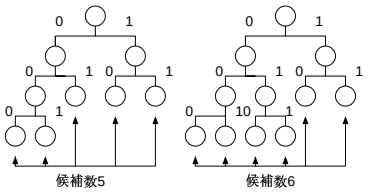

図 5　木構造による候補の有効利用

図 5 は、置き換え候補数が 5 個と 6 個の場合の木構造による bit 列の割り振りを示している。3.1 節から 3.3 節の方法ではどちらも 2bit しか埋め込むことができないが、この方法を利用することで全ての候補を利用でき、候補数に応じた確率で埋め込み量を 1bit 増やすことができる。

4 実験結果とその検証

3 節で挙げられたそれぞれの埋め込み方法を圧縮器 LHA に実装し、その機能の調査を行う。LHA は圧縮法 lh5(辞書サイズ 8KB(Byte)、最大一致長 256) とする。また、テストデータとして、ステゴデータには乱数 bit 列を、元データには Calgary Corpus[2] にある多様な 14 個のデータを使用するものとする。その内、これらの埋め込み方法が効果的に利用できたと考えられる 13 個の元データ (世に一般的に存在するデータ群) に対して検証していく。その他、1 個の元データ (非常にデータ内容が偏ったもの) に対しては実用的に向かないと考えら

れる為、ここでの検証は省く。

4.1 木構造による候補の有効利用と埋め込み量の変化

3.4 節の方法は 3.1 節から 3.3 節までの方法と併用することで埋め込み量がほぼ確実に増えると推測される。ここでは、3.3 節までの方法を実装したステゴ圧縮器 (comprN) と、3.4 節までの方法を実装したステゴ圧縮器 (comprM) による埋め込み量の違いを比較し、3.4 節の方法の効果を検証する。尚、これより検証で示すデータのグラフは全て、取り扱った 14 個のテストデータの内テスト結果が同じ傾向にあるデータ (14 個中 13 個のデータ) の総和である。

図 6 は、comprN と comprM を使用した埋め込み量、そしてこれら 2 つの埋め込み量の差を示した。また、横軸に取る操作長とは、最長一致文字列に対して、どの長さの部分文字列までを候補とするかを決める値である。

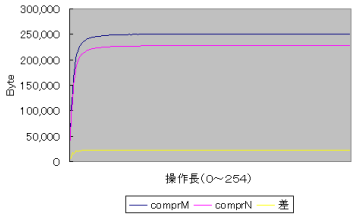

図 6　余り利用時の埋め込み量

図 6 より、常に comprM の方が埋め込み量が多い結果となった。また、3.4 節の方法の有無に関わらず、埋め込み量が著しく変化している部分は操作長 10 程度くらいまでだけで、それ以降はほぼ一定値に収束していることがわかる。

また comprM による埋め込み量の増加量を**図 7** に示す。操作長の範囲は 0 から 9 までとした。

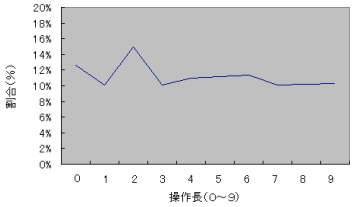

図 7　余り利用時の埋め込み増加量

平均でおよそ 10% の増加が見られた。また、これは操作長に関わらずほぼ一定の結果が得られている為、3.4 節の木構造の適用はどの操作長でも平均的に埋め込み量を増やすのに有効であり、最大埋め込みをする際には効果的であることがわかる。

4.2 最大埋め込み量と圧縮サイズ

ここでは、最大埋め込み量と圧縮サイズとの関係を調査する。検証は comprM にて行う。**図 8** は最大埋め込み量とそれに伴う圧縮サイズの変化 (通常圧縮サイズに対するサイズの膨張量) を示したものである。範囲は、変化の著しい操作長 0 から 9 までとした。

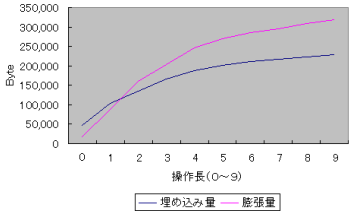

図 8　最大埋め込み量と圧縮サイズの膨張量

操作長が 0 及び 1 では圧縮サイズ膨張量より埋め込み量の方が多く、埋め込みの効率は良いと考えられる。それ以降の操作長ではこれらの操作長と比べ、全て埋め込み量より膨張量の方が多い為、埋め込みの効率が良いとは言えない。また、埋め込み量 1B に対する膨張量 (B) を**図 9** に示す。

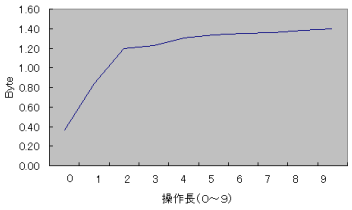

図 9　埋め込み 1Byte に対する圧縮の膨張量

操作長 0 では埋め込み量が 1B 増えるに従って圧縮サイズが 0.4B 膨張しており、操作長 1 では埋め込み 1B に対して 0.7B の膨張、収束値付近では埋め込み 1B で 1.5B の膨張が見られる。

埋め込み量以上の膨張が見られても、圧縮後のサイズに対する膨張の度合が目に見えて増加するものでなければ埋め込みは効果的であると言える。ここで、収束値に到る付近までの操作長の、埋め込み無し圧縮サイズに対する埋め込み有り圧縮サイズの増加量の度合を**図 10** に示す。

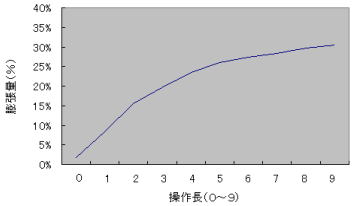

図 10　通常圧縮サイズに対する膨張の度合

最も埋め込み効率の良いと考えられる操作長 0 及び 1 における膨張はそれぞれ、5%、10% 以下で収束値では約 30% の膨張が見られる。埋め込み量と同様、一致長を操作しない場合と値が収束する付近の操作長では膨張量も 5 倍近くなっている。埋め込み量に対する膨張の度

合だけ見ると効率が良いのは操作長 0 及び 1 である。最大量埋め込んでも膨張は 30% で済んでおり、元データよりも大きくなっているわけではないのでステガノグラフィの観点から見れば問題は無い。

また、元データサイズに対してどれだけの埋め込みが可能であるかを**図 11** に示す。横軸に操作長、縦軸に元データサイズに対する埋め込み量の割合としている。

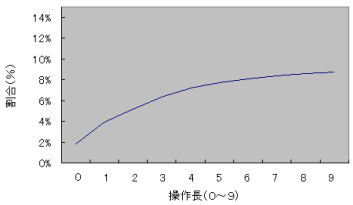

図 11　元データに対する埋め込み量

最大埋め込み量は元データサイズの約 10% までとなった。また、操作長 0 及び 1 では元データの 5% 以下の埋め込みとなり最大埋め込み量の半分以下となっている。

4.3　実行時間

埋め込み機能を実装したことで、最長一致文字列以下の文字列も探索する必要がある為辞書内で文字列を探索する作業が増え、それに伴って実行時間に変化が出てくると予測される。機能の実装により、目に見えてステゴ圧縮 (復号) 時間が掛かってしまう様では実用に向かない。また、ステゴ圧縮データを一般復号器で復号した場合にも同様の事が言える。テストデータ 13 個について、ステゴ圧縮 (復号) 時の実行時間をそれぞれ**図 12** に示す。

尚、実行環境は OS:WindowsXP Home Edition(5.1, ビルド 2600) CPU:Intel Pentium4 2.40GHz メモリ:1024MB RAM, java:version 1.5.0-bata2 である。グラフは横軸を操作長、縦軸を実行時間 (ミリ秒) としている。

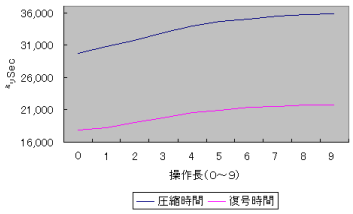

図 12　実行時間 1

これまでの検証と同様にある値で収束を迎えている為、操作長の範囲を 0 から 9 までとした。全ての検証について値がほぼ同一の操作長から収束を迎えていることから、それ以降での一致長の操作による埋め込み作業はさほど効果的ではないと考えられる。また、値が収束するまでの圧縮及び復号時間は操作長に対して線形に近い関係となった為、埋め込み量を増やすに従い実行時間が増えると推測される。また、テストデータ 13 個の圧縮 (復号) 時間が最低約 30(18) 秒、収束値付近でも 35(23) 秒、平均 1 つのテストデータに対して約 3(1.5) 秒の実行時間となっており、実用化の観点から見て許せる範囲である。更に、ステゴ圧縮データを一般復号器で復号した場合の復号時間は、一般圧縮データを復号する場合と比べて特に変化は現れなかった為、問題なく利用できる。

5　おわりに

ステガノグラフィに最も適した埋め込み方法は一致長を最大限操作することで最大量の埋め込みが実現された。ただし、元データが文章やその他、文字列 (ソースプログラムの様な同様の文字列が多数出現する様なデータ) が多いものの場合にはある一定の操作以降は変化が現れなかった。埋め込める最大量については元データサイズの約 10% 程度までとなった。埋め込み効率だけを見ると、操作長 0 及び 1 が、埋め込み 1B に対し膨張サイズが 1B 以下となっている為、最も効率が良い。最大量埋め込みをしない場合は、これらの操作が最も埋め込みの事実に気づかれ難いと考えられる。また実行時間については、同一文字が非常に偏って存在する特殊な元データに対しての埋め込みは不向きであると言える。実用化から見て、これら特殊な元データに埋め込みをしなければ、問題なく運用できると考えられる。

本研究では圧縮器 LHA に重点を置き提案されている埋め込み方法を実装し、その機能の評価と検討を行った。実装した埋め込み方法は全てスライド辞書圧縮部分のみに適用できるものであり、同一アルゴリズムを利用した圧縮器であれば実装可能であると言える。LHA 以外に同一のアルゴリズムを使った圧縮器に対して、同様の機能を適用することでステゴ圧縮データを世に浸透させることができるのではないかと推測できる。

参考文献

[1] 奥田晴彦, 山崎敏 : “LHA と ZIP 圧縮アルゴリズム × プログラミング入門”, SOFTBANK (2003).

[2] “The Standard Calgary Corpus” http://www.data-compression.info/Corpora/CalgaryCorpus.zip.

[3] 儘田 真吾: “データ圧縮符号器での秘匿情報埋め込み”, 2002 年電子情報通信学会ソサイエティ大会, A-6-3 (2002).

[4] 儘田 真吾: “データ圧縮での情報埋め込み”, http://www.isl.cs.gunma-u.ac.jp/~shingo/labo/algo/data/hide-1.pdf.

[5] Mikhail.J.Atallah & Stefano Lonardi: “Authentication of LZ-77 Compressed Data”, Proceedings of the ACM Symposium on Applied Computing (2003).