秘匿データとステゴデータをプログラムとする生成型ステガノグラフィ

2003MT066 永井 義昭　　2003MT102 谷口 喜浩

指導教員 真野 芳久

1. はじめに

秘密通信の一種としてステガノグラフィがある。この技術はインフォメーションハイディングの一分野で、図 1 に示すように、あるメディアをカバーデータとし、その中に隠したいデータである秘匿データを埋め込み、ステゴデータとして通信することで、通信当事者以外の第三者から秘匿データの存在を気付かれないことを目的とした技術である。

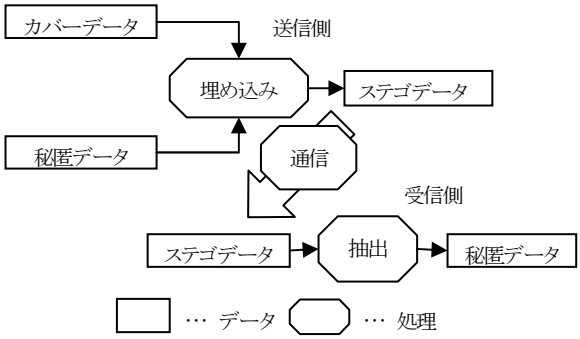

図 1　ステガノグラフィの流れ

現状でステガノグラフィは、情報の埋め込み効率のよさからデジタル画像や音楽データへの埋め込みが主流となっており、攻撃者はそれらのメディアを調べることが多い。そこで、従来取り上げられてこなかった秘匿データをプログラムとし、プログラムソースコードに秘匿データのプログラムを直接埋め込む方法を取る。今回はステゴデータとなるプログラムを生成しながら秘匿データを埋め込むという生成型ステガノグラフィの一手法を提案する。

この手法によって、プログラム中の重要なアルゴリズムなどを隠して通信することができる。生成型とはカバーデータを必要とせず、意味を持つとは限らないプログラムを自動的に作成していく方法である。

2. 生成型ステガノグラフィの概要

秘匿データ、ステゴデータは Java プログラムとして進める。生成型では秘匿データから直接ステゴデータができるので、送信側ではカバーデータを用意する必要はない。ステゴデータを生成する際に秘匿データは分割して埋め込む、この分割されたデータを秘匿断片と呼ぶ。また、ステゴデータは構文レベルでのコンパイルエラーを起こさないように検討を進めていく。秘匿断片を埋め込むために、位置情報ステゴデータと秘匿断片ステゴデータを生成する。秘匿断片ステゴデータには秘匿断片を並び替えて埋め込まれており、位置情報ステゴデータにはその並び替えの情報（位置情報と呼ぶ）が埋め込まれている。受信側は秘匿断片ステゴデータから秘匿断片を、位置情報ステゴデータからは位置情報を抽出し、秘匿データを復元する。これが本研究における生成型ステガノグラフィの全体的な流れである。図 2 でこれを示す。

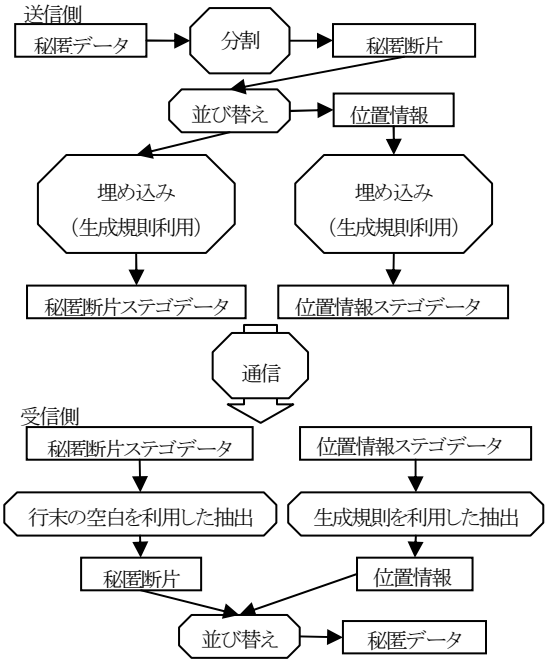

図 2 本研究における生成型ステガノグラフィの流れ

3. 秘匿断片ステゴデータ

3.1 秘匿データの分割

秘匿データは分割して埋め込むので、どのように分割するかを決める必要がある。秘匿データを分割する位置は図 3 のように Java の構文を表した拡張 BNF の右辺に分割記号を置くことによって決める。また、秘匿断片の埋め込みの際には右辺の要素の識別が必要となるので、右辺の終端記号、非終端記号にそれぞれコードを割り当て、右辺の要素を識別できるようにしておく。分割記号は実際には JavaCC のアクションを使って生成規則中に挿入する。

```
IfStatement::="if""("Expression")"¥Statement("else"¥Statement)?
分割記号:¥   397 398   399          400     401     402
```

図 3 分割記号を含む構文の記述例とコード割り当て

3.2 経路情報の取得

生成型でプログラムを生成しながら秘匿断片を埋め込む際に、拡張 BNF からプログラムをどのように生成し、文法的に正しく秘匿断片を埋め込むかの情報が必要である。この情報を経路情報と呼ぶ。経路情報を得るのにまず、JavaCC で秘匿データの構文解析をする際、アクションを使って解析木を作る。JavaCC は LL(1)で解析するので、解析木は下向き構文解析法で生成していく。秘匿データを「1+2*3¥n」とした時の解析木生成の動きと解析木を示すデータを図 4、図 5 に示す。

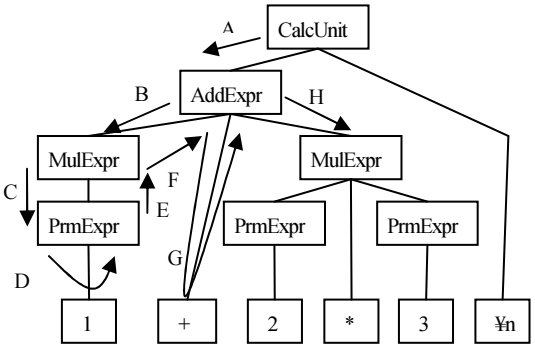

図 4　解析木生成の動き

```
Call:CalcUnit
Call:AddExpr          ―――――――― A
Call:MulExpr          ―――――――― B
Call:PrmExpr          ―――――――― C
Token:Literal:“1”     ―――――――― D
Return:PrmExpr        ―――――――― E
Return:MulExpr        ―――――――― F
Token:“+”             ―――――――― G
Call:MulExpr          ―――――――― H
                      …
```

図 5　解析木の出力結果

ここで図 5 は、構文解析をして、その解析木を出力した結果であり、図 4 でその生成の動きを示している。木は前順で探索し、Call:PrmExpr は生成規則 PrmExpr を呼び出し、Return:PrmExpr は生成規則 PrmExpr から呼び出した生成規則に戻ることを意味している。そして、Token:は拡張 BNF での特定の終端記号を表しており、Token:Literal は数字の 0～9 の数字列を表している。経路情報は、解析木の根から分割記号がある非終端記号までに現れる非終端記号をコード化して列にしたものとし、分割記号までの Call と Return のリスト操作で得ることができる。例えば、秘匿データを「1+」のように分割したいのであれば、図 5 の G のところで分割できるように拡張 BNF 中に分割記号を置く。図 5 で上から一行ずつ見て Call ならばリストに非終端記号を追加し、Return ならばリストの末尾を削除するという操作を G まで行うと、CalcUnit と AddExpr というデータがリストに残り、これが経路情報となる。秘匿断片は G までの終端記号を記録しておけば得ることができる。経路情報と秘匿断片の組を得たら、それをランダムに並び替える。

3.3 秘匿断片の埋め込み

ステゴデータの生成は、拡張 BNF の非終端記号に対応したメソッドを作ることにより行う。メソッドはそれぞれの拡張 BNF の右辺の非終端記号を呼び出す処理や、終端記号を出力する処理を行う。埋め込みは必ずルートのメソッドから始まり、ルートのメソッドは経路情報を参照し、経路情報にあたる非終端記号のメソッドには引数として1を渡し、それ以外の非終端記号のメソッドには引数として 0 を渡す。引数が 1 のメソッドは経路情報から適切な導出を行い、引数が 0 のメソッドについては導出を完了する適当な処理を行う。これを、経路情報の最後の非終端記号のメソッドに1の引数を渡すまで行う。この埋め込みの流れは図 6 のようになる。

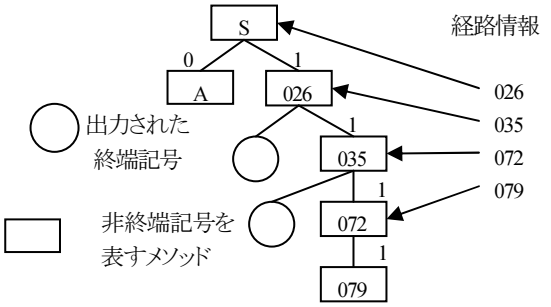

図 6　経路情報から秘匿断片埋め込みの流れ

図 6 で、ルートである S のメソッドは経路情報を参照すると 026 のコードを得るので 026 のメソッドに引数 1 を渡し、A のメソッドは経路情報とは関係ないので、引数 0 を渡し導出を完了し、これを秘匿断片の埋め込みの処理を行う 079 のメソッドまで行なっている。

経路情報の最後のメソッドが引数1ならば、そのメソッドはまず秘匿断片を表すコードから、秘匿断片が右辺のどの要素にあたるかを判断し、秘匿断片にあたらない右辺の要素は適当に導出を完了し、秘匿断片にあたる要素では秘匿断片の出力の処理をする。この方法では、一つの秘匿断片を埋め込むのに必ず同じルートの非終端記号から始まるので、実際の秘匿断片ステゴデータでは一つのクラスに一つの秘匿断片しか埋め込むことができないという問題が出てくる。

3.4 秘匿断片の抽出

秘匿断片ステゴデータから行末の空白を利用し秘匿断片を抽出する。利用の仕方は行末に空白が現れたら、現れた行から抽出を開始し、次の行末に空白が現れるまで抽出し続ける。秘匿断片が一行だけの場合は、行末に 2 個の空白を付加する。

4. 位置情報ステゴデータ

4.1 位置情報の埋め込み

4.1.1 符号化を利用した二進表現への変換

位置情報は0を含む数の並びで表現されている。それを、二進表現し符号化すると符号の終点を知ることができないので位置情報全てに 1 をプラスする。また、位置情報は可変長である。従って、位置情報の終点を知る必要がある。そこで新たな符号化を考えた。その符号化は位置情報の二進表現の前にその桁分の 0 を挿入し位置情報の最後に 1 を挿入する、つまり、γ符号より 0 がひと桁多く最後に 1 が挿入された符号化である。例えば図 7 のような符号化である。

位置情報	2、3、0、1
位置情報に 1 プラスした二進表現	11、100、1、10
用いた符号	0011,000100,01,0010,1
符号化した二進表現	00110001000100101

図 7 符号による位置情報生成の流れ

4.1.2 位置情報の埋め込み

式の文法の例を用い、位置情報ステゴデータ生成の考え方を述べる。そして、この考え方を Java の文法に応用して位置情報を埋め込んでいく。足し算と掛け算からなる式の拡張BNFを表すと図 8 のようになる。それを埋め込むビットと対応を付けるために、図 8 ではなく図 9 の単純な拡張 BNF を使っている。つまり、繰り返し回数を高々1 回に限定した拡張BNFを使い図 9 のようにビットを対応させる。

```
E := T( “+” T)*
T := F( “*” F)*
F := id | “(” E “)”
```

図 8 式の拡張BNF

E := T	0	E := T “+” T	1
T := F	0	T := F “+” F	1
F := id	0	F:= “(” E “)”	1

図 9 拡張BNFとビットの対応

位置情報を埋め込む方法は、E から導出を開始し、図9の位置情報とビット情報の対応にそって導出を行っていく。また、抽出時に正しく位置情報を抽出するためには導出先に非終端記号が二つ現れたときには右の非終端記号から埋め込んでいく。なぜなら、抽出プログラムをJavaCCで記述し、そのアクションを生成規則の右側に記述するため、非終端記号が分割されている場合は深いアクションから現れ、連結されている場合は左の生成規則のアクションから現れる。他にも、位置情報を全て埋め込む前に全て終端記号に導出された場合は新たな開始記号から埋め込みを開始する。この場合は抽出する際に必ず後から生成された開始記号から抽出を行う。位置情報が埋め込み終わったときに、非終端記号が残っている場合は、全て終端記号となる最短経路をたどる。この埋め込みを「2413」を符号化したビット列に行った例を表1に示す。

表 1 位置情報埋め込みの例

動作	状態	情報	式
E:=T	T	0	
T:=F	F	0	
F:=“(”E“)”	(E)	1	
E:=T	(T)	0	
T:=F	(F)	0	
F:=id	(id)	0	(id)
E:=T	T	0	(id)
T:=F“＊”F	F“＊”F	1	(id)
…	…	…	…
F:=id	id+id*(id)	0	(id),id*id,id*id,id+id*(id)

この考えをJava へ応用し位置情報ステゴデータを埋め込んでいく。応用する際に式の文法と異なる点は、導出の種類が 3 つ以上の可能性があるということと宣言部など意味レベルでのコンパイルエラーが起こりうる箇所に対しては情報を埋め込まないという2 点である。3 種類以上の場合には対応するビット情報の種類を増やす。宣言部、変数名などは位置情報の埋め込みとは別に後からコンパイルエラーの起こらないように変換する。

4.2 位置情報の抽出

抽出にはJavaCCを利用する。ここでも式の文法の例を用い、位置情報抽出の考え方を説明し、その考え方を Java へ応用する。抽出するときに、E からの導出先が「T」か「T+T」のどちらかを判断しなければならないので、生成規則の最後に、導出先に対応したビット情報を抽出するアクションを付加する。そのアクションによって位置情報を抽出する。その例を図 10 に示す。そして、抽出されたビット情報を逆順に出力することによって位置情報を得ることができる。

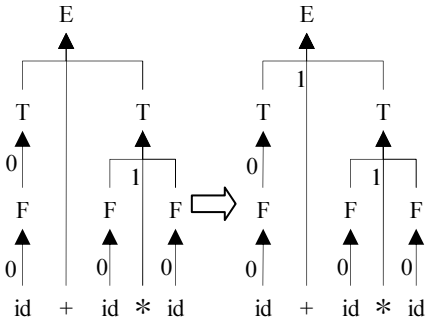

図 10　位置情報抽出の例

この考えをJavaへ応用し位置情報を抽出する。応用する際に式の文法と異なる点は導出の先が3種類以上である可能性があることと、埋め込みの際にビット情報を埋め込まなかった導出に対してはアクションを付加しないという 2 点だけであり、導出が3種類以上の場合には抽出のアクションを書く際にも逆順に記述しなければならない。

4.3 秘匿データの復元

抽出された秘匿断片と位置情報を順に対応付けて、その秘匿断片に対応した数字の順番に並び替えてできたデータが秘匿データである。

5. 評価実験

秘匿断片ステゴデータと位置情報ステゴデータの生成手法についての評価を行う。

5.1 秘匿断片ステゴデータ生成の評価

秘匿データの埋め込み結果は表2のようになり、秘匿データに対し秘匿断片ステゴデータは約 2 倍の大きさになることがわかった。一つの終端記号からなる秘匿断片を埋め込むのにも一つのクラスを生成しているので埋め込み効率は悪いと言えるが、一つの終端記号からなる秘匿断片はコメントを使い、まとめて埋め込むといった方法を取ることにより、埋め込み効率を上げることは可能である。

表 2　秘匿データの埋め込み結果

秘匿データ	秘匿断片数	秘匿断片ステゴデータ
121 byte	15	589 byte
3.02 KB	123	6.36 KB
9.4 KB	411	21.7 KB

5.2 位置情報ステゴデータ生成の評価

ステガノグラフィでは埋め込み効率が重要となる。位置情報と位置情報の桁数に応じてできた位置情報ステゴデータの行数を表 3 に示す。桁数が増えると桁数の 2 倍前後の byte 数で埋め込むことが可能であるといえる。位置情報を埋め込む際に、まだ、全ての生成規則を使っていないのでさらに埋め込み効率を上げることは可能である。

表 3　位置情報ステゴデータの埋め込み効率

位置情報	符号化したときの桁数	行数	ファイルサイズ(byte)
01234	24	27	279
012345	30	39	335
0123456	36	37	327
0～31	272	122	760
以下略	1000	435	2390
	5000	2162	11300
	10000	3567	18400

6. おわりに

生成型を利用することによって、妥当なファイルサイズで秘匿断片ステゴデータを生成することができた。

意味解析レベルでコンパイルエラーが起こることの問題解決に関しては、記号表を作り、識別子の名前、型、アクセス範囲を記録してステゴデータを生成するなどのさらなる工夫が必要である。また、提案した秘匿断片ステゴデータ生成においては解析木の根からの経路情報を使って埋め込むので、一つのクラスに一つの秘匿断片しか埋め込むことができず、見た目も不自然になるという問題や、攻撃者が秘匿断片の抽出手法に気づいた場合、容易に抽出できてしまうという問題もある。このような問題を踏まえ、攻撃者に疑われないような埋め込み方法の検討が必要であり、位置情報ステゴデータ生成に関しては、生成規則全体を利用していないのでシステムの改良の余地がある。

参考文献

[1] 滝澤修, 山村明弘, 牧野京子: テキスト秘密分散法の研究,情報通信研究機構季報,Vol.51 Nos.1/2, pp.171-180 (2005).

[2] 青木一憲, 伊久美茂雄, 河田喬仁:ソースコードに対するステガノグラフィの検討と実現, 南山大学数理情報学部情報通信学科 卒業論文 (2006.1).

[3] 五月女健治: JavaCC コンパイラ・コンパイラ for Java, テクノプレス (2003).

[4] 中田育男: コンパイラ, 産業図書株式会社(1981.9).

[5] 中川裕志, 滝沢修, 井上信吾: ドキュメントへのインフォメーションハイディング, 情報処理 44 巻 3 号 , pp.248-253 (2003.3).

[6] 松本勉: インフォメーションハイディングの概要, 情報処理 44 巻 3 号, pp.227-235 (2003.3).

[7] 滝沢修, 松本勉, 中川裕志, 村瀬一郎, 牧野京子: 改行位置を利用したテキストステガノグラフィ, 情報処理学会論文誌, Vol.45 No.8, pp.1977-1979 (2004.8).