Javaクラスファイルに対する電子透かし法

門田 暁人[†1] 松本 健一[†1] 飯田 元[†2]
井上 克郎[†1,†3] 鳥居 宏次[†4]

本論文では，盗用の疑いのあるJavaプログラムの発見を容易にすることを目的として，プログラム著作者の署名等を電子透かしとしてJavaクラスファイルに挿入する方法，および，取り出す方法を提案する．提案方法は，透かしを挿入してもプログラムの実行効率が変わらず，ツールによる透かしの取り出しの自動化が可能であり，ツールを公開した場合にも透かしの上書き攻撃を回避できるという特徴を持つ．透かしの耐性を実験により評価した結果，obfuscatorと呼ばれるプログラムの難読化ツールによる攻撃後も電子透かしは消えないこと，および，逆コンパイル，再コンパイルによる攻撃に対しても半数以上の透かしは消えないことが確認された．

A Digital Watermarking Method for Java Class Files

Akito Monden,[†1] Ken-ichi Matsumoto,[†1] Hajimu Iida,[†1]
Katsuro Inoue[†1,†3] and Koji Torii[†4]

The aim of this paper is to easily identify an illegal Java program containing a stolen class file. This paper proposes a method for encoding and decoding a digital watermark, such as a program developer's copyright notation, into/from Java class files. Characteristics of the proposed method are: Execution efficiency of the target program is not reduced by watermark encoding, watermark decoding can be automated, and, an overwriting attack is prevented even if a watermark encoding tool is distributed in the public. The result of the experiment to evaluate our method showed that all the watermarks embedded in class files survived an obfuscator attack, and also showed that more than half of watermarks survived a decompile-recompile attack.

1. はじめに

現在，多くのプラットフォーム上で広く流通しているJavaの実行プログラム（Javaアプレット，および，Javaアプリケーション）は，ユーザや第三者によって盗用されやすいという問題をかかえている[20)~23)]．ここでいう盗用とは，既開発プログラムの一部または全体を，正当な権利なしに新規開発プログラムに組み込み，販売・配布することである．

Javaの実行プログラムは，クラスファイルという再利用性の高いプログラム部品に容易に分解できる．しかも，クラスビューワやクラスエディタを用いることで，クラスの継承関係，メソッド名，型，引数等を知ることができる[5),27)]．また，Mocha[30)]，SourceAgain[2)]，SourceTec Java Decompiler[28)]等の逆コンパイラを使うことで，クラスファイルからソースコードを復元できる場合もある．ソースコードが完全に復元されず，再度コンパイルできない場合でも，得られた不完全なコードから，クラスファイルの機能を分析し，盗用することは可能である[8)]．現状では，「あらゆるプラットフォームで動作する」というJavaの特性を生かしたまま，アルゴリズム，クラスのインタフェース，変数の型等の情報を隠蔽し，盗用を防止することは困難である．

盗用防止の技術に代わって注目されているのが，盗用の事実を立証する技術である[10),14)]．盗用の立証が容易になれば，盗用を実質的に抑止できる．ただし，Javaプログラムの著作者にとって，盗用の疑いのあるプログラム（被疑プログラム）を発見することがそも

†1 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology
†2 奈良先端科学技術大学院大学情報科学センター
Graduate School of Information Science, Nara Institute of Science and Technology
†3 大阪大学大学院基礎工学研究科
Graduate School of Engineering Science, Osaka University
†4 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

そも容易でない．特に，Javaプログラム中の一部のクラスファイルだけが盗用された場合，プログラムを実行しただけでは盗用クラスファイルを含むかどうかは必ずしも分からない．また，現在数多く流通しているプログラム難読化ツール（Obfuscator[1),9),26)]）が盗用後のプログラムに適用された場合，プログラムの表現が変わってしまうため，プログラムを解析しても，盗用クラスファイルを特定することは容易でない[19)]．現状では，Java クラスファイルが盗用されていたとしても，そのことに気づくことすら難しい．

本論文では，被疑プログラムの発見を容易にすることを目的として，著作者の署名等の文字列を，電子透かしとしてクラスファイルに挿入する方法，および，取り出す方法を提案する．提案方法の特徴は次のとおりである．

- 実際には実行されないメソッド（ダミーメソッド）をクラスファイルに追加し，ダミーメソッド中に透かしを挿入する．したがって，透かしを挿入しても，プログラムの時間的な実行効率はほとんど変わらない．
- Obfuscator による変換の影響を受けにくい部分（スタック操作命令等）に透かしを挿入する．したがって，Obfuscator 適用後も透かしの大部分を取り出すことができる．
- メソッド単位で透かしを挿入するため，透かしの取り出し処理はダミーメソッドの先頭から行えばよい．したがって，透かしを挿入した位置をプログラム中に記録したり，プログラム解析によって特定したりする必要がない．なお，透かしを暗号化しておけば，どのメソッドからの取り出し結果が実際の透かしであるのかは，第三者には容易に判断できない．
- 提案方法に基づく電子透かし挿入ツール，および，取り出しツールを Web ページで安全に公開することができる[24)]．一般に，電子透かしツールを公開すると，透かしの上書き攻撃☆を回避するのが難しくなる．しかし，提案方法では，透かしを挿入するたびに，ダミーメソッド（透かしが挿入される場所）を追加するため，透かしが上書きされることはない．

プログラムに対する電子透かし法はこれまでにもいくつか提案されている．ただし，その多くは，Java クラスファイルの盗用において被疑プログラムを発見するという目的には適していない．たとえば，文献 4), 15) では，プログラム実行時に特定の入力を与えると電子透かしが出力されるよう，対象プログラム全体を変更する方式が提案されている．この方式では，クラスファイルのようなプログラム中の一部（部品）だけが盗用された場合，透かしが消える，もしくは，取り出しが困難となる．文献 7), 11) では，対象プログラムへのダミー命令の追加，命令の並べ替え等により電子透かしを挿入する方式が提案されている．ただし，透かしを取り出すためには，透かしを挿入した位置をまず特定する必要があり，被疑プログラム発見のコストを大きくする．また，実行効率の低下，透かしの上書き攻撃や Obfuscator 等のプログラム変換に対する耐性について必ずしも考慮されていない．

以降，2 章では，Java プログラム盗用への対策の現状と問題点を列挙する．3 章では，電子透かしの有用性について整理する．4 章では，提案する電子透かし法について述べる．5 章では，提案方法の評価実験とその結果について述べる．6 章では，関連研究を紹介する．7 章では，本論文で得られた結果と今後の課題を簡単にまとめる．

2. Java プログラム盗用への対策

2.1 NET コンパイラによる方法

NET（Native Executable Translation）コンパイラ☆☆は，Java ソースプログラムを計算機依存のバイナリプログラムに変換する[12)]．得られたバイナリプログラムは，Java クラスファイルを含まないので，プログラムが部品単位で盗用される危険性はほぼなくなる．現在，Symantec Visual Cafe，Microsoft Visual J++，Asymetrix SuperCede 等，数多くの NET コンパイラが普及している．

しかし，得られたバイナリプログラムは，Microsoft Windows 等の特定の環境でしか動作させることができず，「あらゆるプラットフォームで動作する」という Java の特長が失われる．また，NET コンパイラを用いて，Java アプレットを作成することもできない．

2.2 Obfuscator による方法

Obfuscator は，クラスファイル中のクラス名，メソッド名，変数名等のシンボルを無意味な文字列に変換するため，逆コンパイル後のソースコードの解析を困難にできる．現在，SourceGuard[1)]，Jshrink[9)]，DashO[26)]等，数多くの Obfuscator が普及している．

☆ 透かし挿入済みのプログラムに対して透かし挿入ツールを適用することで，透かしを上書きしてしまう攻撃[13)]．

☆☆ スタティックコンパイラ，AOT（Ahead-of-time）コンパイラとも呼ばれる．

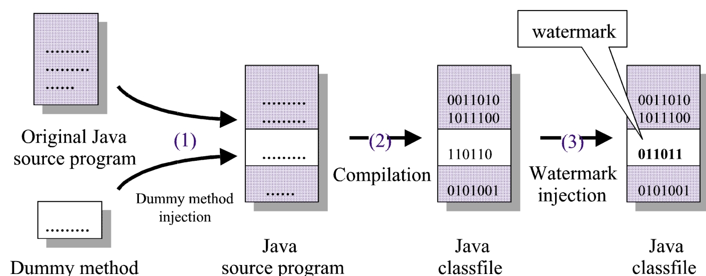

図 1 電子透かし挿入手順
Fig. 1 Overview of watermark encoding procedure.

しかし，Obfuscator では，アルゴリズム，クラスのインタフェース，変数の型等の情報を隠蔽することはできない．

2.3 getCodeBase による方法

getCodeBase()，getDocumentBase() 等の命令を用いると，特定の Web サーバに置いた場合にのみ動作するアプレットを作成することができる[25]．これにより，ダウンロードされたアプレットが，他の Web サーバ上で無断使用されることを防止できる．

しかし，この方法は Java アプリケーションには適用できない．また，逆コンパイルによりソースコードが得られた場合，ソースコーの改変により，getCodeBase()，getDocumentBase() を比較的容易に無効化できる[25]．

3. 電子透かしの有用性

著作者の署名等の文字列を，電子透かしとしてあらかじめ各クラスファイルに挿入しておき，電子透かしが容易に取り出せるような仕組みを設けておけば，被疑プログラムの発見を容易にできる．画像や音声を対象としたものであるが，電子透かしを利用してインターネット上での著作権侵害を自動的に発見するサービスが実際に提供されている[16]．Javaプログラム，特に，Java アプレットは，インターネットの Web ページ上に置かれるため，電子透かしとネットワーク探索ロボットを併用することで，アプレット中の盗用クラスファイルを容易に特定できる可能性がある．

実用上の観点からは，クラスファイル中の電子透かしは，次の性質を満たすことが重要である．

- 透かしを挿入しても，プログラムの時間的な実行効率は変わらない．
- 透かしの取り出しを自動化（ツール化）できる．
- Obfuscator 適用後も透かしを取り出すことができる．
- 透かしの上書き攻撃に耐性がある（電子透かし電子透かし挿入ツールを販売，配布しても，透かしの安全性が保たれる）．

4. 電子透かし法

4.1 電子透かしを挿入する方法

提案する電子透かし挿入法は，3 つの手順から構成される（**図 1** 参照）．

（手順 1）ダミーメソッドの挿入

コンパイル前のソースプログラムに対して，実際には実行されないメソッド（ダミーメソッド）を追加する．ダミーメソッドに含まれるプログラムコードは，透かしの文字列を書き込むための領域となる．ダミーメソッドの追加は，透かし挿入のたびに行う．したがって，透かしの挿入によって，すでに存在する透かしが上書きされることはない．

次に，ダミーメソッド呼び出し文を，プログラム中に追加する．ダミーメソッド呼び出し文の例を以下に示す．

```
if (Expression) Dummy_Method();
```

‘Expression’ には，つねに偽となる式を記述する．したがって，ダミーメソッドが実際に実行されることはない．式の記述時の注意点として，プログラム盗用者が逆コンパイルによってソースコードを入手する可能性があるため，‘Expression’ はある程度複雑な（たとえば配列変数の参照を含むような）式とし，真となる場合がないとは容易に判定できないようにすることが望ましい．プログラムの盗用者は，プログラムの実行によって ‘Expression’ がほとんどの場合に真とはならないことに気づく可能性があるが，一般的なプログラムには例外処理を開始するかどうかを判定する条件文が多く含まれ，そこでは，ほとんどの場合に真とはならない式が用いられている．したがって，含まれる式が真となる場合を発見できないというだけの理由でダミーメソッド呼び出し文が特定されるとは限らない．

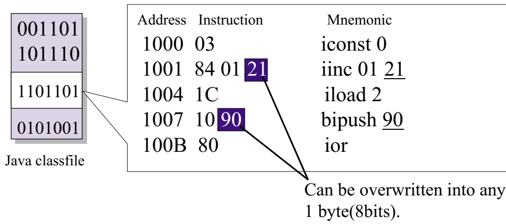

図 2 数値オペランドの書換え箇所の例
Fig. 2 Overwriting numerical operands.

Instruction	assigned bits
60 iadd	… 000
64 isub	… 001
68 imul	… 010
6C idiv	… 011
70 irem	… 100
7E iand	… 101
80 ior	… 110
82 ixor	… 111

Rule 1

Instruction	assigned bits
9B iflt	… 00
9C ifge	… 01
9D ifgt	… 10
9E ifle	… 11

Rule 2

Instruction	assigned bits
C6 inull	… 0
C7 inonnull	… 1

Rule 3

図 3 オペコードに対するビット割当て規則の例
Fig. 3 Example of bit assignment rules for opcodes.

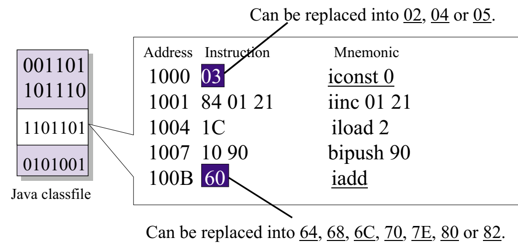

図 4 オペコードの置換え箇所の例
Fig. 4 Replacing opcodes.

なお，ダミーメソッド呼び出し文を実行頻度が高くない部分に挿入すれば，プログラム実行効率の低下は避けられる．

（手順 2）コンパイル

ダミーメソッドを追加した Java ソースコードをコンパイルし，クラスファイルを生成する．

（手順 3）電子透かしの挿入

クラスファイル中のダミーメソッドの先頭から，電子透かしとなる文字列の書き込みを行う．ただし，Java の実行形式の 1 つである Java アプレットは，その実行の直前に文法や型をバイトコード検証器によってチェックされる[18]．透かしを書き込む際には，文法の正しさや型の整合性を保つ必要がある．具体的には，次に示す書換え可能な特定の数値オペランドとオペコードのみを透かしの挿入箇所とする．

(a) 数値オペランドの書換え

4 種類のオペコード *bipush*, *sipush*, *iinc*, *wide iinc* は，その数値オペランドの値を書き換えても文法の正しさと型の整合性が保たれる．数値オペランド書換えの箇所の例を**図 2** に示す．この例では，第 1 ローカル変数に 21 を足す *iinc 01 21*，および，スタックに 90 を積む *bipush 90* の各数値オペランド（*21* および *90*）の書換えが可能である．数値オペランドの書換えにより部分的にはプログラムの仕様が変化するが，書換えはダミーメソッド内に限定されるためプログラム全体としての仕様は保存される．

なお，その他のオペコードでは，文法の正しさを保ちながら数値オペランドを任意の値に書き換えることはできない．たとえば，クラステーブルの操作に関するオペコード *getfield* や *putfield* の数値オペランドを書き換えると，クラステーブルに不整合が生じ，アプレットが実行できなくなる．

(b) オペコードの置換え

オペコードの中には，相互に置換しても文法の正しさと型の整合性が保たれるものがある．たとえば，スタック内の要素の足し算を行うオペコード *iadd* は，引き算やかけ算等の演算を行うオペコード *isub*, *imul*, *idiv*, *irem*, *iand*, *ior*, *ixor* に置き換えることができる．つまり，*iadd*, *isub*, *imul*, *idiv*, *irem*, *iand*, *ior*, *ixor* の 8 個のオペコードは互いに可換であるといえる．この性質を利用して，バイトコード中の *iadd* を，可換な 8 個のオペコードのいずれかに置き換えることで，3 ビットの情報を埋め込むことにする．たとえば，*iadd* を 000_2，*isub* を 001_2，*imul* を 010_2，$\cdots$，*ixor* を 111_2 にそれぞれ割り当てることで，000_2〜111_2 の情報が表現できる．

スタックの演算を行うオペコード以外にも，互いに可換なオペコード群が存在する．Java バイトコード全体では，16 個のオペコード群が存在し，合計 25 ビットの情報を割り当てることが可能である．オペコード群に対するビット割当て規則の例を**図 3** に示す．また，オペコード置換え箇所の例を**図 4** に示す．この例では，スタックに 0 を積むオペコード *iconst 0* は，可換な 4 個のオペコードのいずれかに置き換えることで 2 ビットの情報を埋め込むことができる．同様に，オペコード *iadd* には 3 ビットの情報の埋め込みが可能である．

透かしの挿入は，まず，透かしとする文字列をビット列に変換することから始まる．そして，ダミーメソッドの先頭から透かし挿入箇所（書換え可能な数値オペランドとオペコード）を検索し，透かしとなるビット列を先頭から順に書き込んでいく．数値オペランド部分には 8 ビット，オペコード部分にはビット列

割当て規則に従って 1〜3 ビット分の透かしを書き込む．“(C)AKITO MONDEN” という文字列をバイトコードに挿入した例を**図 5** に示す．この例では，書換え可能な 2 つの数値オペランドと 2 つのオペコードを利用して，合計 21 ビットの透かしが挿入されている．

数値オペランドとオペコードには，メソッド名や変数名といったシンボル情報が含まれていない．したがって，Obfuscator による変換で透かしが上書きされたり消されたりする可能性は低い．

4.2　電子透かしを取り出す方法

各メソッドの先頭から透かし挿入箇所（書換え可能な数値オペランドとオペコード）を検索し，書き込まれているビット列を順に取り出し，文字列に再構成することで，透かしを取り出すことができる．**図 6** の例では，method4 がダミーメソッドで “(C)AKITO MONDEN 1999” という透かしが取り出されている．例からも分かるように，ダミーメソッド以外のメソッドからの取り出し結果は無意味な文字列となる．また，プログラム解析等によりダミーメソッドを特定する必要がなく，透かし取り出し処理の自動化（ツール化）が容易となっている．

なお，書換え可能な数値オペランドやオペコードのうちで実際に透かし挿入箇所として利用するもの，オペコードに対するビット割当て規則，透かしとする文字列からビット列への変換規則等を非公開とすることで，第三者による透かしの取り出しを困難にすることができる．また，著作者の署名等の文字列を暗号化したうえで透かしとして挿入しておけば，どのメソッドからの取り出し結果が実際の透かしであるのかは，第三者には容易に判断できない．

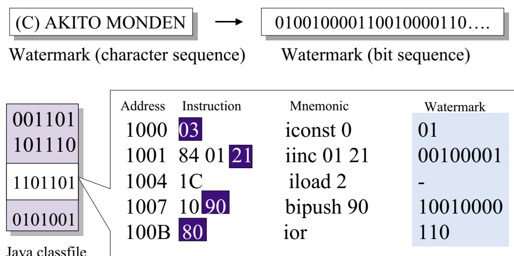

図 5　電子透かしの挿入例
Fig. 5　Example of watermark encoding.

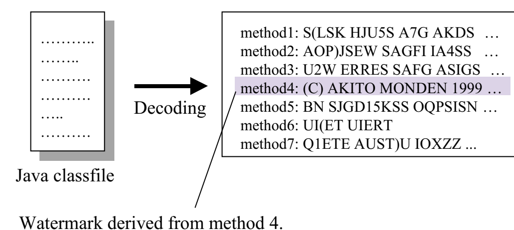

図 6　電子透かし取り出しの例
Fig. 6　Example of watermark decoding.

5. 評価実験

我々は，電子透かし挿入ツール，および，取り出しツールを作成し，Web 上で公開している[24]．これらのツールを用いて挿入できる透かしの耐性を実験により調べた．

プログラムの盗用者は，現在流通している種々の Java プログラム変換ツールを用いて，プログラム中の電子透かしの消去を試みる可能性がある．本論文では，**図 7** に示すように，(1) Obfuscator による攻撃，(2) 逆コンパイル，再コンパイルによる攻撃の 2 通りの方法に対して，電子透かしの耐性の評価を行った．

5.1　実験手順

（手順 1）クラスファイルの準備

透かし挿入対象として，JDK1.2 に付属しているサンプルのクラスファイルから無作為に 10 個のクラスファイルを選んだ．透かし挿入のためには各クラスファイルにダミーメソッドを追加しておく必要があるが，ここでは簡単のため，クラスファイル中の既存のメソッドを仮のダミーメソッドと見なした．なお，10 個のクラスファイルに含まれるメソッド（合計 96 個）のうち，透かしを挿入するのに十分なサイズを持ったメソッドは合計 23 個であった．それらの 23 個のメソッドを，透かしを書き込むためのダミーメソッドと見なすことにした．

（手順 2）電子透かしの挿入

23 個のダミーメソッドそれぞれについて，文字列 “(C)AKITO MONDEN” を挿入した．

（手順 3）クラスファイルに対する攻撃

電子透かし挿入後のクラスファイルに対して，次の 2 通りの攻撃を加えた．

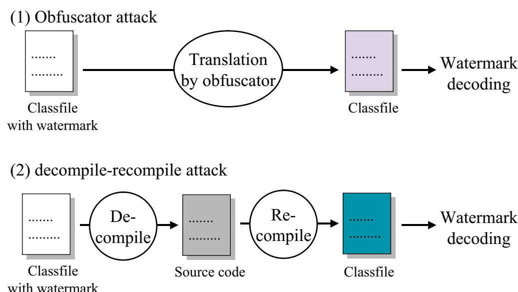

図 7　クラスファイルに対する 2 通りの攻撃
Fig. 7　Two kinds of attacks against class files.

(1) Obfuscatorによる攻撃

強力なobfuscatorの1つとして広く用いられている4thpass社製Source Guard 2.0[1]を使い，電子透かし挿入後の各クラスファイルを難読化した．

(2) 逆コンパイル，再コンパイルによる攻撃

広く用いられている逆コンパイラの1つであるMocha[30]を用いて，電子透かし挿入後の各クラスファイルからソースファイルを復元した．さらに，復元されたソースファイルに対して，JDK1.2のjavacを用いてコンパイルを行い，再びクラスファイルを導出した．コンパイル時には，最適化オプションを用いた．

（手順4）電子透かしの取り出し

攻撃を加えた後のクラスファイルから，電子透かしの取り出しを試みた．

5.2 実験結果

(1) Obfuscatorによる攻撃結果

Obfuscatorによる攻撃を加えた後の23個のメソッドからは，すべて正しく電子透かしを取り出すことができた．obfuscatorの多くは，メソッド名やローカル変数名等のシンボル情報の変換を行うが，メソッド中のオペランド，オペコードにはほとんど影響を与えないためであると考えられる．

(2) 逆コンパイル，再コンパイルによる攻撃結果

図8に示すように，攻撃前の時点では，10個のクラスファイル中に合計23個の電子透かしが挿入されている．逆コンパイルの結果，5個のクラスファイルは逆コンパイルに失敗した（逆コンパイラの停止）．逆コンパイルできた残りの5個のクラスファイルについては，クラスファイル中の文法誤りを人手により修正することで，すべて再コンパイルに成功した．再コンパイル後の5個のクラスファイルに対して透かしの取り出し処理を行ったところ，もとから含まれていた8個の電子透かしのうち，正しく電子透かしを取り出すことができたのは5個であった．逆にいえば，逆コンパイル，再コンパイル攻撃によって消すことのできた電子透かしは，23個中わずか3個であった．

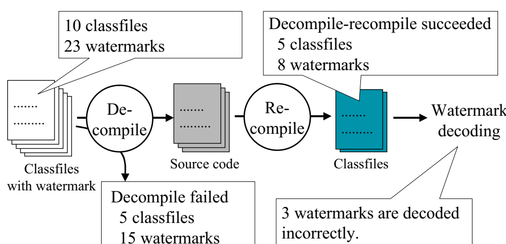

図8 逆コンパイル，再コンパイルに対する実験結果
Fig. 8 Result of decompile-recompile attack.

以上の結果より，(1) Mochaを用いた逆コンパイル，再コンパイルは必ずしも成功しない，(2) 成功した場合にも，半数以上（8個のうち5個）の透かしが消えずに残っていることが分かった．1個のクラスファイル中に複数個の電子透かしを挿入しておくことで，逆コンパイル，再コンパイル攻撃に対しては実用上十分な耐性を持つ．

6. 関連する研究

6.1 プログラムに対する電子透かし法

従来，画像データ，音声データ，テキスト文書等の著作物に電子透かしを挿入する方法が，さかんに研究されてきた[3),6)]．一方，プログラムに対する電子透かし法も，少数ではあるが，近年になっていくつか提案されている[4),7),11),15)]．ただし，これらのいずれの方法も，被疑プログラムの発見を容易にすることを目的としていない．あらかじめ各ユーザの識別子を電子透かしとして各プログラムに挿入しておくことで，違法コピープログラムの流通が発覚した場合に，違法コピーを行ったユーザを特定することを目的としている．そのため，Javaクラスファイルのようにプログラム内の部品が盗用された場合の発見を容易にするという目的には向いていない．

Collbergらは，プログラム実行時に特定の入力を与えると電子透かしが出力されるよう，対象プログラム全体を変更する方法を提案している[4)]．また，北川らは，Javaプログラム中の配列変数に透かしを挿入し，プログラムの実行時に取り出す方法を提案している[15)]．これらの方法では，透かしの取り出し時には，プログラム全体が存在し，かつ，実行できることが前提となっている．そのため，クラスファイルのようなプログラム部品が盗用された場合には，透かしが消えてしまう，あるいは，取り出しが困難となる．

廣瀬らは，C言語のソースプログラムに対して任意のビット列を電子透かしとして挿入する方法を提案している[11)]．この方法では，透かしの挿入対象となるプログラムに対して，変数宣言の順序を入れ換える，ダミーの変数宣言を追加する等の変更を加えることで，ビット列が挿入できる．同様の方法として，Davidsonらは，実行プログラム（バイナリプログラム）中の各命令コードを並べ替える方法を提案している[7)]．ただし，これらの方法では，透かしを取り出すためには，取り出し処理を開始する位置をまず特定する必要がある．プログラムの一部分だけが盗用された場合には，必ずしも自動的に透かしが取り出せない．また，すでに透かしが挿入されているプログラムに対して，さら

に故意に別の透かしを挿入すると，透かしが上書きされてしまうという問題がある．Obfuscatorや逆コンパイル等のプログラム変換に対する耐性についても，必ずしも考慮されていない．

6.2 署名付きJavaアプレット

署名付きアプレットの概念は，プログラムの開発者等の情報をプログラムに付加するという点では，電子透かしと似ている．Javaセキュリティモデルは，署名付きアプレットによる認証のメカニズムを提供しており，ユーザは暗号化された署名をアプレットと組み合わせて復号することで，アプレットが第三者によって改変されていないオリジナルのものであることを保証する[17),29)]．

しかし，アプレットの署名は，アプレットに含まれるクラスファイルの盗用を防ぐという目的には役立たない．アプレットの署名はクラスファイル中に埋め込まれているのではなく，クラスファイルと独立に存在しているため，クラスファイルの盗用者は，容易に署名をとりはずすことができる．署名によりプログラムの盗用を抑止するためには，署名とクラスファイルが容易に分離できないこと，すなわち，電子透かしとして署名がクラスファイル中に埋め込まれていることが必要となる．

7. まとめと今後の課題

本論文では，Javaプログラム部品の盗用を容易に検知するために，Javaクラスファイルに対して，任意の文字列を電子透かしとして挿入する方法，および，取り出す方法を提案した．透かしの耐性を実験により評価した結果，obfuscatorによる攻撃後もメソッド中の電子透かしは消えないこと，および，逆コンパイル，再コンパイルによる攻撃に対しても半数以上の透かしは消えないことが確認された．

従来研究では，プログラムに対する電子透かしの原理については論じられているが，それを実際に応用することの必要性や実用性については，必ずしも具体的に整理されていない．

これに対し，本論文では，「盗用されたJavaクラスファイルの発見」という実用的な目的を設定し，それに対する解として妥当な性質を持った電子透かし法の提案を行った．提案方法を用いることで，たとえば，インターネット上のアプレット等に対して透かしの有無を検知・検査するサーチロボットを稼働させて，プログラムの盗用を監視することが現実的に可能となる．

さらに従来研究では，提案された方式の実装，評価は必ずしも行われていないのに対し，我々は，本提案方法に基づいて実装した電子透かしツールをWebページ上で公開し，実際に利用・評価を行えるように無償提供している[24)]．本論文では，限定的ではあるが，提案方法による電子透かしの耐性についても実装したツールを用いて評価を行っている．

提案方法も含めて，現状では，あらゆる攻撃に対して耐性のある電子透かし法は存在しない．盗用者が容易に行える攻撃であるObfuscatorによる攻撃に対しては，提案方法は強い耐性を持つ．しかし，より強い攻撃に対しては耐性を持つとは限らない．たとえば，プログラムを逆コンパイルし，ソースコードの大部分を人手により書き換えるといった攻撃を防ぐことは，現状ではどのような電子透かし法でも困難である．今後は，より強い攻撃に対しての耐性を高めるために，電子透かし法に符号誤り訂正技術を応用することについて検討していく予定である．

謝辞 本研究の遂行にあたって貴重な助言をいただいた，電子技術総合研究所の一杉裕志氏に深く感謝いたします．

参考文献

1) 4thpass LLC: SourceGuard. http://www.4thpass.com/
2) Ahpah Software: SourceAgain. http://www.ahpah.com/sourceagain/
3) Berghel, H.: Watermarking cyberspace, *Comm. ACM*, Vol.40, No.11, pp.19–24 (1997).
4) Collberg, C. and Thomborson, C.: Software watermarking: Model and dynamic embeddings, *26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (*POPL'99*), San Antonio, Texas (Jan. 1999).
5) Comware Australia Pty. Ltd.: ClassNavigator. http://www.comware.com.au/classnavigator/classnav.htm
6) Craver, S., Memon, N., Yeo, B. and Yeung, M.M.: Resolving rightful ownerships with invisible watermarking techniques: Limitations, attacks, and implications, *IEEE Journal on Selected Areas in Communications*, Vol.16, No.4, pp.573–586 (1998).
7) Davidson, R.L. and Myhrvold, N.: Method and system for generating and auditing a signature for a computer program. *US Patent 5,559,884*, Assignee: Microsoft Corporation (Sep. 1996).
8) Dyer, D.: Java decompilers compared, in article of JavaWorld (July 1997). http://www.javaworld.com/javaworld/jw-07-1997/jw-07-decompilers.html

9) Eastridge Technology: Jshrink.
http://www.e-t.com/jshrink.html
10) 古屋栄男，松下 正，眞島宏明，田川幸一：知って得するソフトウェア特許・著作権 改訂版，アスキー (1997).
11) 廣瀬直人，岡本英司，満保雅浩：ソフトウェア保護に関する一考察，1998 年暗号と情報セキュリティシンポジウム，SCIS '98-9.2.C (1998).
12) Hsieh, C.A., Conte, M.T., Johnson, T.L., Gyllenhaal, J.C. and Hwu, W.W.: Optimizing NET compilers for improved Java performance, *Computer*, Vol.30, No.6, pp.67–75 (1997).
13) 一杉裕志：ソフトウエア電子すかしの挿入法，攻撃法，評価法，実装法，情報処理学会夏のプログラミングシンポジウム報告集，pp.57–64 (July 1997).
14) 金井重彦：著作権の基礎知識—マルチメディア時代のコンピュータ・プログラム，ぎょうせい (1998).
15) 北川 隆，楫 勇一，嵩 忠雄：Java で記述されたプログラムに対する電子透かし法，1998 年暗号と情報セキュリティシンポジウム，SCIS '98-9.2.D (Jan. 1998).
16) 丸紅ソリューション：ECOM.
http://senet.msol.co.jp/ecom/
17) 丸山 宏，小島 尚：主に Java 及び Active X におけるコード署名安全性に関する考察，コンピュータソフトウェア，Vol.16, No.4, pp.23–32 (1999).
18) McGraw, G. and Felten, E.: *Java security: hostile applets, holes, and antidotes*, John Wiley & Sons (1997).
19) 門田暁人，高田義広，鳥居宏次：ループを含むプログラムを難読化する方法の提案，電子情報通信学会論文誌（D-I），Vol.J80-D-I, No.7, pp.644–652 (1997).
20) 門田暁人，飯田 元，松本健一，鳥居宏次，一杉裕志：プログラムに電子透かしを挿入する一手法，1998 年暗号と情報セキュリティシンポジウム，SCIS '98-9.2.A (Jan. 1998).
21) 門田暁人，飯田 元，松本健一，鳥居宏次：Java プログラムを対象とする電子透かし法，日本ソフトウェア科学会第 16 回大会論文集，pp.253–256 (Sep. 1999).
22) Monden, A., Iida, H., Matsumoto, K., Inoue, K. and Torii, K.: Watermarking Java programs, *Proc. 4th International Symposium on Future Software Technology* (*ISFST '99*), Nanjing, China, pp.119–124 (Oct. 1999).
23) Monden, A., Iida, H., Matsumoto, K., Inoue, K. and Torii, K.: A practical method for watermarking Java programs, *Proc. 24th Computer Software and Applications Conference* (*COMPSAC2000*), Taipei, Taiwan (Oct. 2000).
24) 門田暁人：Java watermarking tools.
http://tori.aist-nara.ac.jp/jmark/
25) Nolan, G.: Decompile once, run anywhere: protecting your Java source, *Web Techniques Magazine*, Vol.2, Issue 9 (1997).
26) preEmptive Solutions: DashO.
http://www.preemptive.com/
27) Raud, R.: ClassViewer.
http://raud.net/robert/classinfo/ClassViewer.html
28) SourceTec Software Co.: SourceTec Java Decompiler.
http://www.srctec.com/decompiler.htm
29) Sun Microsystems, Inc.: Security and signed applet.
http://www.javasoft.com/products/jdk/1.1/docs/guide/security/
30) Vliet, H.: Mocha – the Java Decompiler.
http://www.brouhaha.com/~eric/computers/mocha.html

(平成 12 年 4 月 27 日受付)
(平成 12 年 9 月 7 日採録)

門田 暁人（正会員）

平成 6 年名古屋大学工学部電気学科卒業．平成 10 年奈良先端科学技術大学院大学博士後期課程修了．同年同大学情報科学研究科助手．博士（工学）．ソフトウェアの知的財産権の保護，ソフトウェアメトリクス，ヒューマンインタフェース等の研究に従事．日本ソフトウェア科学会会員．

松本 健一（正会員）

昭和 60 年大阪大学基礎工学部情報工学科卒業．平成元年同大学大学院博士課程中退．同年同大学基礎工学部情報工学科助手．平成 5 年奈良先端科学技術大学院大学助教授．工学博士．Computer-Aided Empirical Software Engineering（CAESE）環境，ソフトウェアメトリクス，ソフトウェアプロセス，視線インタフェースに関する研究に従事．電子情報通信学会，IEEE，ACM 各会員．

飯田　　元（正会員）

昭和 63 年大阪大学基礎工学部情報工学科卒業．平成 3 年同大学大学院博士課程中退．同年同大学基礎工学部情報工学科助手．平成 7 年より奈良先端科学技術大学院大学助教授，現在に至る．平成 10 年ドイツ Kaiserslautern 大学および Fraunhofer IESE 客員研究員．平成 10～11 年カナダ British Columbia 大学客員研究員．博士（工学）．ソフトウェア環境，開発プロセス，システム設計法等の研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE，ACM 各会員．

井上　克郎（正会員）

昭和 54 年大阪大学基礎工学部情報工学科卒業．昭和 59 年同大学大学院博士課程修了．同年同大学基礎工学部助手．昭和 59～61 年ハワイ大学マノア校情報工学科助教授．平成元年大阪大学基礎工学部情報工学科講師．平成 3 年同学科助教授．平成 7 年より同学科教授，現在に至る．平成 11 年奈良先端科学技術大学院大学教授（併任）．工学博士．ソフトウェア工学の研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE，ACM 各会員．

鳥居　宏次（正会員）

昭和 37 年大阪大学工学部通信工学科卒業．昭和 42 年同大学大学院博士課程修了．同年電気試験所（現電子技術総合研究所）入所．昭和 59 年大阪大学基礎工学部情報工学科教授．平成 4 年奈良先端科学技術大学院大学教授．現職は同大学副学長．工学博士．専門はソフトウェア工学．IEEE，ACM および情報処理学会の各フェロウ．