クラス構造変換手法を用いたJavaプログラムへの利用者識別情報の埋め込み手法

福島和英[†,☆] 田端利宏[††,☆☆]
田中俊昭[†††] 櫻井幸一[††]

ソフトウェアの盗用を立証するために，利用者の識別情報をプログラムに埋め込むことが考えられる．門田らによって，Java クラスファイルに任意の文字列を埋め込む手法が提案されている．しかし，この手法は，すべてのクラスファイルに同一の情報を埋め込むことを前提としている．このため，この手法を用いて利用者ごとに異なる識別情報を埋め込んだ場合，複数のプログラムを比較することで埋め込み部分がただちに判明してしまう．本研究では Java プログラムのクラス構造を変換することで，電子透かしの埋め込み位置の特定を難しくすることを検討する．

A Software Fingerprinting Scheme for Java Using Class Structure Transformation

Kazuhide Fukushima,[†,☆] Toshihiro Tabata,[††,☆☆]
Toshiaki Tanaka[†††] and Kouichi Sakurai[††]

Embedding personal identifiers as watermarks to software is effective in order to protect copyright of them. Monden et al. proposed program watermarking scheme for embedding arbiter character sequence to target Java class files. But their scheme can be used to embed only the same watermarking to all the programs. Thus, if we apply their scheme to embed users' personal identifiers, the watermark can be specified by comparing two or more users' program. This paper improve the problem by using a class structure transformation.

1. はじめに

集積回路設計技術の向上によって，ハードウェアの開発コストは低くなっている．一方，ソフトウェアの開発コストは，ハードウェアほど低くなっていないのが現状である．計算機性能の向上により複雑な処理を実行することが可能となったことにともない，開発されるソフトウェアも複雑で規模が大きいものになったことが，1つの要因である．また，近年ではソフトウェア開発を支援するフレームワークや再利用技術が普及しているものの，ソフトウェアの開発の根本的な部分は人手に頼る面が大きい点も要因となっている．

ソフトウェアがネットワークを介して広範囲に流通すると，不正者がこのソフトウェアの著作権を侵害する可能性がある．1つの例としては，不正者がソフトウェアを入手し，そのソフトウェアの著作権を不正に主張することが考えられる．もう1つの例としては，不正者がソフトウェアを複製し，インターネットに流通させることが考えられる．

前者の脅威への技術的な対策としては，ソフトウェアに著作権所有者の識別情報を電子透かしとして埋め込む方法がある．著作権主張者は，あらかじめ埋め込んでおいた電子透かしを抽出することにより，自らが正当な著作権所有者であると主張できる．これまでに提案されてきたソフトウェアに対する電子透かし手法の多くは，このことを目的とした手法である．一方，後者の脅威への対策としては，各利用者の識別情報を埋め込む方法がある．この場合，著作権所有者は，不正に流通しているソフトウェアから電子透かしを抽出

† 九州大学大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
†† 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University
††† 株式会社 KDDI 研究所
KDDI R&D Laboratories Inc.
☆ 現在，株式会社 KDDI 研究所
Presently with KDDI R&D Laboratories Inc.
☆☆ 現在，岡山大学大学院自然科学研究科
Presently with Graduate School of Natural Science and Technology, Okayama University

することで，不正にソフトウェアを流通させた利用者を特定することができる．

近年では，広く知られている商用ソフトウェアが不正に流通するケースが多く見受けられる．このため，不正者がソフトウェアの著作権を不正に主張することは少なく，ソフトウェアの不正な流通そのものが大きな問題となっている．すなわち，後者の脅威がより深刻であると考えられる．電子透かしにより，ソフトウェアを不正に流通させた利用者を特定できれば，この利用者に対し，法的な責任を負わせることができる．また，不正行為を行った利用者を特定できることを広く通知することにより，ソフトウェアの不正な流通を抑制することも期待できる．

しかしながら，各利用者の識別情報をソフトウェアに埋め込む電子透かし手法は提案されていないのが現状である．そこで，本論文では，既存の電子透かし手法を拡張し，各利用者の識別情報を Java プログラムに埋め込む手法を提案する．

2. 電子透かし

電子透かしを利用することにより，ソフトウェアの盗用を証明することが可能である．電子透かしは大きく分類して，コンテンツの著作権所有者の識別情報を埋め込むものと，コンテンツの利用者の識別情報を埋め込むものの 2 つに分けられる．

2.1 コンテンツの著作権所有者の識別情報の埋め込み

コンテンツの著作権所有者の識別情報を電子透かしとして埋め込む手法は，主に 2 つの目的がある．

目的 1 コンテンツの正当性を証明する

たとえば，我々は紙幣の透かしの存在を確かめることによって，これが日本銀行によって発行された正当な紙幣であることを確認できる．この場合，透かしはすべての人にとって認識可能なものでなければならない．

目的 2 著作権所有者が自らの著作権を証明する

たとえば，あるコンテンツの著作権所有者がコンテンツ内に自らの識別情報を電子透かしとして埋め込んでおいたとする．このとき，他人がこのコンテンツを入手して，不正に著作権を主張しようとしても，著作権所有者は，あらかじめ埋め込んでおいた自らの識別情報を復元することによって，自分がこのコンテンツの著作権所有者であることを証明することができる．

この場合，電子透かしは，コンテンツの著作権所有者以外の人間にとって，識別可能である必要はない．むしろ，他人が識別できたとすると，電子透かしに対して改ざん，あるいは除去などの攻撃が行われる可能性がある．この場合の電子透かしには，これらの攻撃に対する耐性が要求される．

2.2 コンテンツの利用者の識別情報の埋め込み

利用者の識別情報を電子透かしとして埋め込む手法は，コンテンツの不正な複製および再配布を行った利用者を追跡するために用いることができる．たとえば，コンテンツを利用者に分配する際に，各利用者の識別情報を何らかの電子透かしとしてコンテンツに埋め込んでいたとする．配布されるコンテンツは電子透かし部分が利用者ごとに異なっている．このとき，ある利用者がコンテンツを不正に複製し，再配布したとする．著作権所有者は，流通している不正コンテンツを入手し，識別情報を復元することにより，不正な利用者を特定することができる．埋め込まれた利用者の識別情報は，不正な複製および再配布を行った人物を特定するための情報であるので，これらの情報の改ざんは，前節で述べた開発者の識別情報の改ざんに比べて深刻なものになると考えられる．

2.3 関 連 研 究

従来，画像データ，音声データ，テキスト文書などの著作物に電子透かしを埋め込む手法がさかんに研究されてきた[1),6)]．これらのコンテンツは冗長度が高く，コンテンツ中の数ビットを変更しても，人間には知覚されない場合が多い．このため，比較的簡単に多くの情報量を埋め込むことができる．一般に電子透かしは利用者の攻撃によって容易に消されないことが重要である．画像データや音声データ中の電子透かしも容易には除去することは不可能である．多くの場合，電子透かしを除去する過程において，画像そのものを劣化させてしまうためである．

一方，プログラム言語は画像，音声，一般的なテキスト文章と比較して，冗長度が低い．たとえば，プログラムの 1 ビットを変更しただけでも，正しく動作しなくなることがありうる．このため，効率良く電子透かし情報を埋め込むことは困難であるとされてきた．しかし，近年になって，冗長な命令を挿入したり，可換な命令を入れ替えたりすることによって，プログラムに透かし情報を埋め込む手法が考案されている．

Java を対象とする電子透かし手法としても，さまざまな手法が提案されている[4),5),7),11)~14)]．

北川ら[14)] は，Java のプログラムに対して任意の数値列を電子透かしとして埋め込む手法を提案している．この手法では電子透かしを格納するための変数を用意し，その変数に電子透かしの情報を表す数列を格納す

る．門田ら[13]は，Javaのプログラムの数値オペランド部分，オペコード部分に電子透かしを埋め込む手法を提案している．この手法ではソースコードに対してダミーのメソッドを追加することによって電子透かしを埋め込むための領域を確保する．また，Venkatesanら[12]は，透かし情報を制御フローのグラフを用いて表現し，本来のプログラムの制御フローとマージすることで電子透かしを埋め込む手法を提案している．この手法は，Collbergら[5]によりSandMark[3]のフレームワークで実装されている．以上の3つの手法は静的な電子透かし手法であり，プログラムを実行せずに電子透かしを抽出することが可能である．一方で，動的な電子透かしは，特定の入力を与えてプログラムを実行すると電子透かしが表示されるという手法である．動的な電子透かしとしては，Collbergらによる手法[4]とThomborsonらによる手法[11]が提案されている．

これらの手法はコンテンツの著作権所有者の識別情報を埋め込むことを前提とした電子透かし手法である．しかし，不正な複製および再配布を行った利用者を追跡するためには，利用者の識別情報を電子透かしとして埋め込むことが不可欠である．そこで，我々は利用者ごとに異なる電子透かしをJavaプログラムに埋め込む手法を検討してきた[7]．この手法における電子透かしの埋め込み方法としては門田らの手法を用いた．グラフを用いた手法[5),12)]，および動的な手法[4),11)]は，門田らの手法と比較して，攻撃に対する耐性が高い可能性がある．しかし，門田の手法には，(1) 透かしの埋め込みによる実行効率の低下がない，(2) プログラムを実行せずに透かしが取り出せるため，多数のプログラムの透かしの有無のチェックが容易に行える，(3) プログラムの部品単位（クラスファイル）に透かしを埋め込むことができる，という利点がある．

本論文は，門田らの電子透かし手法の利点を生かしつつ，利用者ごとに異なる識別情報をJavaクラスファイルに埋め込み，追跡性を持たせることを目的とする．さらに，電子透かしとハッシュ値を併用することにより，改ざんされているクラスファイルに対しては，利用者を特定できる可能性を残すと同時に，改ざんされていないクラスファイルからは，低コストかつ確実に利用者を特定することを実現する．このため，3章で門田らの電子透かし手法について説明し，この手法をそのまま用いて利用者の識別情報を埋め込んだ場合の問題点を指摘する．次に，4章でクラス構造変換手法を用いることにより，利用者の識別情報を埋め込んだ場合の安全性を向上することを検討する．

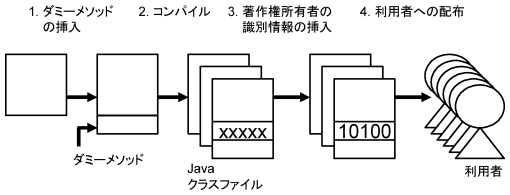

図 1 門田らによる電子透かし手法の概略
Fig. 1 Overview of watermarking scheme by Monden, et al.

3. 門田らによる電子透かし手法

3.1 電子透かしの埋め込み

電子透かしの埋め込みは，以下の4つの手順から構成される．**図 1** に電子透かし手法の概略を示す．

手順 1 電子透かし埋め込み部分の挿入
コンパイル前のソースコードに対して実際には実行されないダミーのメソッドを追加する．ダミーメソッドのソースコードは，電子透かしの文字列を書き込むための領域となる．ダミーメソッドは任意のものでよいが，電子透かし情報を埋め込むために十分なサイズを持つ必要がある．

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，クラスファイルを作成する．

手順 3 電子透かしの埋め込み
クラスファイル中のダミーメソッドに対応する部分に，電子透かしとなる文字列の書き込みを行う．文字列の書き込みにおいては，バイトコード検証器の動作に注意する必要がある．Javaクラスファイルが実行される前には，バイトコード検証器が，プログラムの文法およびデータ型をチェックする．このため，ダミーメソッド中のプログラムコードが文法的に正しく，なおかつ型の整合性が保たれていないと，Javaクラスファイルの実行が拒否される．
文法の正しさおよびデータ型の整合性を保つために，次の2つの手法を併用して電子透かしの埋め込みを行う．
(1) 数値オペランドの書き換え
スタックに値をpushするバイトコード命令の数値オペランド部分，スタック上の値に演算を施すバイトコード命令の数値オペランド部分は任意の数値に置き換えても，文法の正しさや，型の整合性が保たれる．たとえばスタックに値をpushする命令 `bipush x` における数値オペランド `x` は，任意の数値に置き換えることが可能である．これ

表 1 バイトコード命令に対するビット列の割当て例[13)]
Table 1 Example of bit assignment rules for bytecodes.

バイトコード	ニーモニック	割当てビット
0E	dconst_0	0
0F	dconst_1	1
C6 xx xx	ifnull	0
C7 xx xx	ifnonnull	1
9B xx xx	iflt	00
9C xx xx	ifge	01
9D xx xx	ifgt	10
9E xx xx	ifle	11
60	iadd	000
64	isub	001
68	imul	010
6C	idiv	011
70	irem	100
74	iand	101
78	ior	110
7C	ixor	111

により，1 バイトの情報を電子透かしとして埋め込むことが可能である．

(2) バイトコード命令の書き換え

バイトコード命令の置き換えを行うことによっても，電子透かしを埋め込むことは可能である．たとえば，スタック内の 2 つの要素の足し算を行うバイトコード命令 iadd は，引き算や掛け算などの他の演算を行う他のバイトコード命令（isub，imul，idiv，irem，iand，ior，ixor）に置き換えても，文法の正しさおよび型の整合性は保たれる．つまり，iadd，isub，imul，idiv，irem，iand，ior，ixor の 8 個のバイトコード命令は可換であるといえる．この性質を利用して，バイトコード中にこれらのバイトコード命令の中の 1 つが出現した場合は，可換な 8 つのバイトコード命令のうち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ができる．たとえば，iadd を 000，isub を 001，imul を 010 に，...，ixor を 111 に割り当てることによって，000 から 111 までのデータを表現できる．

スタックの演算を行うバイトコード命令以外にも，互いに可換であるバイトコード命令群が存在する．それらのバイトコード命令は，互いに可換なバイトコード命令により，それぞれ 1 ビットから 3 ビットの情報の割当てが可能である．割当て例を，**表 1** に示す．

手順 4 利用者への配布

電子透かしを埋め込んだ Java プログラムを，各利用者に配布する．

3.2 電子透かしの取り出し

電子透かしを取り出すためには，バイトコードと電子透かしのビット列の対応およびビット列と文字の対応を知る必要がある．はじめに，クラスファイルの各メソッドに含まれる数値オペランドとバイトコードを対応するビット列に置き換え，さらに，ビット列を文字列へと置き換える．すると，ダミーメソッドに対応する部分に電子透かしが現れる．

以上の電子透かしの埋め込みおよび取り出しを行うツール jmark[8)] が公開されている．

3.3 問 題 点

門田らの手法を用いて，Java クラスファイルに利用者の識別情報を電子透かしとして埋め込む場合には，複数の利用者の結託攻撃に対する脆弱性が問題となる．著作権所有者の識別情報を電子透かしとして埋め込む場合は，各利用者に配布される Java クラスファイルはすべて同一である．しかし，利用者の識別情報を電子透かしとして埋め込む場合は，クラスファイルの電子透かしの埋め込み部分（ダミーメソッドに対応する部分）が利用者ごとに異なる．このとき，複数の利用者が結託して，各クラスファイルを比較することにより，電子透かしの埋め込み部分が特定される可能性がある．以下に，具体的な攻撃手法を説明する．

2 人の利用者 Alice と Bob が結託し，クラスファイルに埋め込まれた電子透かしを攻撃しようとすると仮定する．ここで，Alice に配布されたクラスファイル TEST0_A.class には，電子透かしとして文字列 Alice が，Bob に配布されたクラスファイル TEST0_B.class には，文字列 Bob が埋め込まれていたとする．

この場合は，Alice と Bob による結託攻撃が可能である．この攻撃手法は 3 つの手順から構成される．

手順 1 逆コンパイル

逆コンパイラを用いて，Java クラスファイルから Java ソースコードを得る．この場合は，Alice のクラスファイル TEST0_A.class からはソースコード TEST0_A.java が得られ，Bob のクラスファイル TEST0_B.class からソースコード TEST0_B.java が得られる．

手順 2 ソースコードの差分の調査

ここでは，埋め込まれた電子透かしが，利用者ごとに異なっていることを利用する．Unix のコマンドである diff を利用し，Alice のクラスファイルから得られたソースコードと，Bob のクラスファイルから得られたソースコードの差分を調べる．

```
% diff TEST0_A.java TEST0_B.java
```

異なっている部分が，電子透かしが埋め込まれた

ダミーメソッドである．

手順 3 電子透かしの除去あるいは改ざん

手順 2 により電子透かしを埋め込まれたダミーメソッドを特定できる．このため，電子透かしに対して除去あるいは改ざんを行うことができる．ダミーコードは実際に呼び出されることはないため，除去あるいは改ざんを行ってもプログラムの実行には影響を与えない．

(1) 電子透かしの除去

Java ソースコードから，上記の過程で見つかったダミーメソッドの定義部分を消去する．その後，コンパイルを行い，再びクラスファイルに変換する．

(2) 電子透かしの改ざん（クラスファイルに対して）

同じ電子透かし埋め込み手法を用いて，攻撃対象とする Java クラスファイルに直接，別の電子透かしを埋め込む．`jmark` は，対象とするクラスファイル，電子透かしを埋め込むメソッドおよび埋め込む文字列の 3 つを引数としてとる．ここでは，電子透かしを埋め込むメソッドとして，手順 2 で見つかったダミーメソッド名を与える．

(3) 電子透かしの改ざん（ソースコードに対して）

Java ソースコードから，上記の過程で見つかったダミーメソッドの定義部分を改ざんする．電子透かしは，バイトコード命令の数値オペランド，オペコード部分に埋め込まれているため，ソースコード中でそれらに対応する部分，すなわち，数値データ，算術演算子を改ざんする．最後に改ざんを行ったソースコードを再コンパイルし，再びクラスファイルに変換する．

以上の手順により，2 人の利用者が結託をして，攻撃を行うことが可能である．手順 1 では，逆コンパイルを行っているが，実用規模のプログラムでは，逆コンパイルにより完全なソースコードを得るのが難しい場合もある．この場合は，逆アセンブラを用いることが考えられる．逆アセンブラは逆コンパイラと比較し，安定して動作する．このため，得られた 2 つのアセンブルコードを `diff` コマンドで比較することで，差分を容易に調査できる．

4. クラス構造変換手法

門田らの手法を用いて，利用者ごとに異なる識別情報を電子透かしとして埋め込む場合には，複数の利用者による結託攻撃が問題となる．各利用者に配布されるクラスファイルの構造が同一であり，電子透かしが埋め込まれているダミーメソッドが同じ位置に存在していることが要因である．

ここで，クラス構造変換手法とは，プログラムの仕様を保ちつつクラスの構造を変換する手法である．同様に，プログラムを読みにくくすることを目的として，プログラムの構造を変換する難読化という手法がある．Java プログラムに対する難読化手法としては，Sakabe らの手法[10]，Chan らの手法[2]，Sosonkin らの手法[9] がある．Sakabe らは，Java ソースコードにメソッドオーバロードおよびインタフェースを導入し，ソースコードに対する静的解析を難しくする手法を提案している．Chan らは，Java バイトコードに含まれる識別子の情報を書き換えることによって，逆コンパイルおよび再コンパイルを難しくする手法を提案している．Sosonkin らは，Java ソースコード中で定義されているクラスの構造を合併，分割により変換したり，インタフェースを利用して変数の型を隠蔽したりする難読化手法を提案している．今回利用するクラス構造変換手法は，クラスに含まれるメソッドを他のクラスに移動することにより，クラスの構造を変換する手法である．

以下で説明するクラス構造変換手法を適用することにより，配布するクラスファイルの構造を利用者ごとに異なるようにすることができ，結託攻撃を難しくすることができる．

4.1 変換手順

Java では，データ構造（フィールド）とそれに対する操作（メソッド）をクラスとしてひとまとめに定義する．さらに，データ構造やクラスの内部処理を行うメソッドに対し，アクセス制限を施すことにより，これらを外部から隠蔽する．このことを，カプセル化と呼ぶ．ここで，説明するクラス構造変換手法では，各クラスのメソッドを，他のクラスに移動することで，Java プログラムを構成する各クラスの構造を変換する．以下に，メソッドを移動する手順を以下に示す．

手順 1 対象とするメソッドの選択

任意のクラスから他のクラスに移動するメソッドを選ぶ．ただし，`main` メソッド，およびそのクラスのコンストラクタ以外のメソッドを選ぶものとする．ここで，選んだメソッドを `m` と呼ぶ．

手順 2 メソッドのクラスメソッド化

手順 1 で選んだメソッド `m` の修飾子を `static public` に設定する．このことにより，メソッド `m` はクラスメソッドに変更され，他のクラスに移動しても，移動先のクラスのインスタンスを作成することなく呼び出すことが可能になる．

手順 3 フィールドの公開

メソッド m を他のクラスに移動すると，このメソッドの操作対象となるフィールドに，外部のクラスから参照・代入を行う必要がある．このため，メソッド m から，参照または代入が行われているフィールドの修飾子を public に設定する．このことにより，外部のクラスに対しこれらのフィールドが公開され，参照または代入を行うことができるようになる．

手順 4 引数の追加

手順 2 で，元々インスタンスメソッドであったメソッドが，クラスメソッドに変更される場合がある．この場合には，このメソッドに引数を追加する必要がある．インスタンスメソッドは，(インスタンス名).(メソッド名) の形式で呼び出されるため，操作の対象となるインスタンスは明確である．一方，クラスメソッドは，クラスそのものに属するメソッドであるため，操作の対象となるインスタンスの情報を引数としてメソッド m に渡す必要がある．

手順 5 メソッドの移動

メソッド m を任意のクラスに移動する．この際，メソッド m を配置するために新たなクラスを定義してもよい．

手順 6 呼び出し側の修整

手順 4 でメソッド m に引数が追加され，さらに，手順 5 で他のクラスに移動したことにより，メソッド m が呼び出されている箇所にも修正を施す必要がある．

以上の手順 1 から手順 6 を繰り返すことにより，メソッドを 1 つずつ他のクラスに移動することができる．このことにより，クラス構造が順次変換されていく．

さらに，フィールド，およびメソッドの識別子を無意味なものに変更することで，クラス構造変換に対する逆変換を難しくすることができる．たとえば，すべてのクラスのメソッド名を順番に a, b, c, ... とリネームすることができる．同じ名前のメソッド名が複数存在していたとしても，配置されているクラスが異なっているため，これらのメソッドを一意に区別することは可能である．しかし，一方でソースコードの可読性は低下するため，逆変換の操作が難しいものとなる．

4.2 具 体 例

図 2 のソースコードが示すクラス A について，クラス構造変換手法の適用例を示す．ここでは，メソッド A2 を他のクラス B に移動するものとする（手順 1）．

はじめに，メソッド A2 の修飾子を public static に設定する．このことにより，メソッド A2 はクラスメソッドに変更される（手順 2）．

```
class A{
   public int a;
   A(){...}
   private void A1(){...}
   public void A2(int x){
      a = x;
   }
   public void A3(){...}
}

class Sample{
   public static void main(String[] args){
      A Obj = new A();
      Obj.A2(3);
      ...
   }
}
```

図 2 クラス構造変換手法を適用する前のソースコード
Fig. 2 Source code before applying class structure transformation.

次に，メソッド A2 はクラス A のフィールド a に対し代入を行っている．このため，フィールド a の修飾子を public に設定する．このことにより，フィールド a に対し，外部のメソッドから代入が行えるようになる（手順 3）．

次にメソッド A2 に，クラス A のインスタンス Obj を引数として追加する（手順 4）．この引数 Obj はクラスメソッドに変更されたメソッド A2 に，操作を行うべきインスタンスを示すために用いられる．さらに，メソッド内部でのフィールド a への代入命令に対しても，インスタンス Obj のフィールド a への代入であることを示す必要がある．

手順 2 から手順 4 までの操作により，メソッド A2 をクラスメソッドとして，他のクラスに移動する準備が完了したことになる．ここで，メソッド A2 をクラス B に移動する（手順 5）．

最後に，メソッド A2 の呼び出し側の修正を行う（手順 6）．図 2 のソースコードで，メソッド A2 はクラス A のインスタンス Obj を介して，Obj.A2(3) の形で呼び出されていた．しかし，本手法を適用することにより，このメソッドには，引数としてクラス A のインスタンス Obj が追加され，さらに，他のクラス B に移動している．ゆえに，B.A2(3,Obj) の形でメソッド A2

```
class A{
   public int a;
   A(){...}
   private void A1(){...}
   public void A3(){...}
}

class B{
   B(){...}
   public static void A2(int x, A Obj){
      Obj.a = x;
   }
}

class Sample{
   public static void main(String[] args){
      A Obj = new A();
      B.A2(3,Obj);
      ...
   }
}
```

図 3 クラス構造変換手法を適用した後のソースコード
Fig. 3 Source code after applying class structure transformation.

が呼び出されるように，呼び出し側を修整する必要がある．

クラス構造変換手法を適用した後のソースコードを**図 3** に示す．また，以上の操作によるクラス構造の変化を示すと，**図 4** のようになる．

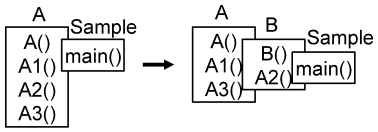

図 4 クラス構造の変化
Fig. 4 Change of the class structure.

5. 提 案 手 法

5.1 電子透かしの埋め込み

電子透かしの埋め込みは，以下の 5 つの手順から構成される．**図 5** に電子透かし手法の概略を示す．

手順 1 電子透かし埋め込み部分の挿入
ソースコードのいずれかのクラス内にダミーのメソッドを挿入する．ダミーメソッドに含まれるプログラムコードは，電子透かしの文字列を書き込むため，十分なサイズを持つ必要がある．

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，クラスファイルを作成する．

手順 3 電子透かしの埋め込み
クラスファイル中のダミーメソッドに対応する部分に，利用者ごとに異なる文字列を識別情報として埋め込む．ここでは，門田らの手法[13] を用いて利用者の識別情報を埋め込む．

手順 4 クラス構造変換手法の適用
クラスファイルに対して，前章のクラス構造変換手法を適用する．
ここで，各利用者のクラス構造がそれぞれ異なるようにメソッドの移動を行う．たとえば，対象となるプログラムで定義されているクラスを $\texttt{Class}_1$，$\texttt{Class}_2$，...，$\texttt{Class}_m$，4 章のクラス構造変換手法で移動可能なメソッドを $\texttt{method}_1$，$\texttt{method}_2$，...，$\texttt{method}_n$ とする．さらに，N 人の利用者（ただし，$N \leq m^n$ とする）の識別番号 ID には，1 以上 N 以下のそれぞれ異なる整数が用いられているとする．このとき，利用者の識別番号 ID を n 桁の m 進数で表し，$\texttt{ID=(ID}_1\texttt{, ID}_2\texttt{, ..., ID}_n\texttt{)}$ とする．最後に，識別情報が ID である利用者のプログラムに対し，各メソッド $\texttt{method}_i$ が $\texttt{Class}_{\texttt{ID}_i+1}$ に配置されるようにクラス構造変換手法を施す．また，電子透かしの取り出しのため，著作権所有者は，利用者の識別情報，透かしを埋め込んだクラスのハッシュ値，および透かしを埋め込んだダミーメソッドの名前の組 (ID, h, method) を利用者ごとに，自らが管理するファイルに登録する．

手順 5 利用者への配布
電子透かしを埋め込んだ Java プログラムを各利用者に配布する．

5.2 電子透かしの取り出し

提案手法により埋め込まれた電子透かしを取り出すためには，ダミーメソッドが配置されたクラス，バイトコードと電子透かしのビット列の対応およびビット列と文字の対応を知っておく必要がある．はじめに，電子透かしの検証者は，不正に流通している Java プログラムを構成するすべてのクラスファイルのハッシュ値を計算する．次に，これらのハッシュ値のどれかと，一致する登録情報を探す．たとえば，これらのハッシュ値の 1 つと h_{A} が一致する利用者 A の登録情報 $(\mathrm{ID_A}, h_{\mathrm{A}}, \mathrm{method_A})$ があったとする．このとき，検証者はハッシュ値 h_{A} を持つクラスファイルから，利用者 A を仮定して，電子透かしの抽出を行う．具体的には，ダミーメソッド $\mathrm{method_A}$ に含まれる数値オ

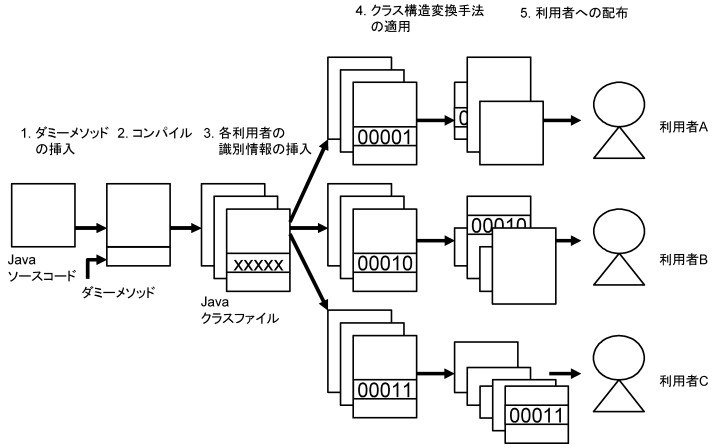

図 5 提案手法の概略
Fig. 5 Overview of proposed method.

ペランドとバイトコードを対応するビット列に置き換え，さらに，ビット列を文字列へと置き換える．得られた文字列と登録情報 ID_A が一致すれば，不正に流通しているプログラムは，利用者 A に配布されたものであると断定することができる．この場合は，1 つのメソッドのみから透かしの抽出を行うため，抽出される文字列は 1 つである．このため，抽出された透かしが複数の利用者の登録情報に合致することはない．

一方で，不正な利用者は埋め込まれている電子透かしを改ざん，あるいは除去することを目的として，配布されたクラスファイルに対し，前節で説明したクラス構造変換手法を適用することが考えられる．この場合は，一致する登録情報を見つけ，この登録情報に対応する利用者を仮定して，電子透かしを抽出することは難しいため，すべてのクラスファイルに含まれるすべてのメソッドから電子透かしの抽出を行う必要がある．ただし，jmark[8] を使った場合，高速に電子透かしが抽出でき（1 回の抽出作業に要する時間は 0.1 秒以下），Java プログラムに含まれるメソッドの総数は多くても数百程度であると考えられるため，すべてのメソッドから電子透かしを抽出することは，現実的な時間で実行可能であると考えられる．この場合は，すべてのメソッドから電子透かしを抽出するので，抽出された複数の文字列が，複数の登録情報に合致する可能性がある．この場合は，合致した複数の登録情報から透かしから正しい登録情報を一意に特定することは不可能である．しかし，このような状況に陥る確率を軽減することは可能である．利用者の総数を N 人，メソッドの総数を n 個，透かしのビット数を $|w|$ としたとき，ある利用者のクラスファイルから，他の利用者の透かしが抽出される確率は，$nN/2^{|w|}$ 以下となる．このため，長いビット数の透かしを埋め込めば，複数の透かしを抽出する場合にも，他の利用者の透かしと合致する確率は低くなる．ただし，長いビット数の透かしを埋め込むためには，ダミーメソッドのサイズを大きくする必要がある．

最後に電子透かしとハッシュ値を併用する理由を説明する．提案手法の電子透かしのみを用いた場合，特定の利用者を仮定した透かしの抽出ができないため，すべてのメソッドから透かしを抽出する必要がある．このため，クラスファイルが改ざんされていない場合でも，透かしの抽出コストが大きくなる．そこで，電子透かしとハッシュ値を併用することにより，改ざんされているクラスファイルに対しては，利用者を特定できる可能性を残すことができ，改ざんされていないクラスファイルからは低コストで透かしを抽出することができる．

5.3 考　　察

以下に提案手法で識別情報を埋め込んだプログラムに対する考察を行う．

5.3.1 提案手法を適用したプログラムの実行効率

利用者の識別情報を埋め込むダミーのメソッドは実行されないので実行時間に影響を与えない．また，クラス構造変換手法もメソッドを配置するクラスのみを変換する手法であるため，実行時間はほとんど増大しないと考えられる．実際にいくつかのプログラムを作成し，クラス構造変換手法を適用する前，および適用した後の実行時間を比較する実験を行った．実験環境の CPU は Intel Pentium III 1 GHz，メモリは 512 MB，OS は Redhat Linux 9 である．**表 2** が示すように，実行時間の増加は 2.4%から 8.2%にとどまった．

表 2 クラス構造変換によるプログラム実行時間の変化
Table 2 Change of the execution times.

	クラス構造変換の適用前				クラス構造変換の適用後			
プログラム名	行数	クラス数	実行時間 [ms]	サイズ [KB]	行数	クラス数	実行時間 [ms]	サイズ [KB]
BinTree	201	3	5,640	3.46	227	14	6,100	4.34
BinSearch	99	2	3,960	1.76	108	11	4,250	2.18
HuffEnc	226	3	6,090	3.83	280	17	6,290	4.76
TopSort	156	3	2,870	2.51	187	10	2,940	2.71
Heapsort	203	3	2,890	3.48	228	12	3,010	4.34

5.3.2 提案手法の利点

提案手法で，クラス構造変換手法を適用することは，以下の 2 つの利点がある．

利点 1 クラスファイルの盗用が難しくなる

Java では，クラスは特定の機能を実現するインスタンスの雛形である．クラスを記述するクラスファイルには，その機能を実現するためのデータ構造とそれに対する操作を定義してある．このため，不正者は，他人が作成したクラスファイルを盗用することにより，自分のプログラムに機能を追加することができる．

しかし，クラス構造変換手法を適用すると，1 つの機能を実現するためのデータ構造とそれに対する操作がクラス間に分離される．このとき 1 つのクラスに着目すると，従来は別のクラスで定義されていた複数のメソッドが散在していることになる．このため，不正者が，提案手法により電子透かしを埋め込まれたプログラムから盗用したクラスファイルをそのまま他のプログラムに流用することはできなくなる．

利点 2 複数の利用者による結託攻撃が難しくなる

門田らの手法を用いて利用者の識別情報を埋め込んだ場合，各利用者に配布されるプログラムを構成するクラスファイルの構造は同一である．このため，攻撃者が異なる利用者に配布されたクラスファイルを逆コンパイルし，それぞれのソースコードに含まれるメソッドを順次比較していく．異なるメソッドが見つかれば，そのメソッドがダミーメソッドである．ソースコードに含まれるメソッドの総数を n とすると，この攻撃が要する比較回数は最大で n 回である．なお，3.3 節で説明したように，この攻撃は，`diff` コマンドを 1 回利用するだけで実行することが可能である．

一方で，提案手法を用いて各利用者の識別情報を埋め込んだ場合，攻撃者は以下の手順でダミーメソッドを特定する必要がある．(1) はじめに，攻撃者は，異なる利用者に配布された複数の Java プログラムを入手する．ここでは，攻撃者が異なる 2 人の Java プログラムを入手したと仮定して，説明を行う．(2) 次に，それぞれのプログラムについて，プログラムを構成するすべてのクラスファイルを逆コンパイルすることによって，Java ソースコードを得る．(3) それぞれの Java ソースコードから，すべてのメソッドを抜き出す．(4) 次に，それぞれのソースコードから得られたメソッドどうしの対応付けを行う．クラス構造変換手法は，あるクラスに属するメソッドを他のクラスに移動することで，クラスの構造を変える手法である．このため，片方のソースコードに存在しているメソッドと本質的に同じ（メソッド名，修飾子，引数などの形式は，クラス構造変換の手順で変更されていることもありうる）メソッドが他方のソースコードにも存在しているはずである．攻撃者は，このような本質的に同じメソッドの組を列挙していく．ただし，攻撃者は 2 つのメソッドの動作が本質的に等しいか否かを，判定できる必要がある．この作業を続けると，最後に対応付けができない（すなわち，本質的に異なる）メソッドの組が残る．これらのメソッドが，それぞれのソースコードに挿入されたダミーコードである．

メソッドの対応付けを行うための計算量について考察する．上と同様に，それぞれのソースコードに n 個のメソッドが含まれているとする．このとき，片方のソースコードに存在する 1 つのメソッドと対応する他方のソースコードのメソッドを見つけるためには，最大 n 回の比較が必要となる．このため，すべてのメソッドに対し，対応する他方のソースコードのメソッドを見つけるためには，最大 n^2 回の比較が必要である．

もし，メソッドを何らかの基準に従いソートできれば，n 回の比較で，メソッド間の対応付けを行うことができる．この場合，メソッドのソートに要する計算量が $n \log n$ であり，かつ比較回数が n 回であるので，攻撃全体に必要となる計算量は $n \log n$ となる．

たとえば，クラスファイルを逆アセンブルして得

られるプログラムから，jmark により変更される可能性のあるオペランド（iadd，isub など）だけをメソッドごとに選び出す．次に，各メソッドから選び出したオペランド集合に基づいてメソッドをソートする．クラスファイル変換手法は，メソッドに含まれるオペランドそのものを大幅に変更する手法ではないので，この方法でメソッドのソートによる攻撃が行われる可能性がある．ただし，ダミーの変数代入や演算を含める難読化手法を併用することにより，この攻撃に対する耐性が増すと考えられる．

以上により，門田らの手法を用いて，利用者の識別情報を埋め込む場合は，電子透かしの挿入箇所の特定に要する比較回数は，Java プログラムに含まれるメソッドの総数に比例する程度に収まる．しかも，`diff` コマンドを利用することにより，1 回のコマンド実行により特定が可能である．しかし，提案手法では，電子透かしの挿入箇所の特定に要する比較回数は，Java プログラムに含まれるメソッドの総数の 2 乗に比例する．しかも，比較の際には，プログラムの本質的な動作が等しいか否かを判定する必要があり，計算機などを用いて自動的に行うのは難しいと考えられる．以上により，提案手法は，規模が大きいプログラムに対してより有効であるといえる．

5.3.3 提案手法の限界

以下に，提案手法の限界について説明する．門田らの方法により電子透かしを埋め込んだ場合，数値オペランドが明らかに不自然になるという問題がある．たとえば，`for(i=0;i<10;i++)` のようなループ文の数値オペランド，およびオペコードが変更されて，`for(i=-1;i<-23;i+=24)` になる場合がある．この場合はプログラム中の `for` 文を調べることで，ダミーメソッドである可能性が高いメソッドを選び出すことができる．このため，`for` 文には透かしを埋め込まない，といった配慮が必要である．

また，各メソッドの数値オペランドやオペコードの分布を調べることでも，電子透かしの入ったダミーメソッドを推測することは可能である．電子透かしの埋め込みには，数値オペランドおよび特定のオペコードが用いられるため，ダミーメソッドにおけるこれらの出現頻度は他のメソッドのものと比較して高くなると考えられる．このため，1 個のメソッドにあまり大きな透かしを埋め込まないようにする，といった配慮が必要である．

6. ま と め

本研究では，既存の電子透かし埋め込み手法を拡張し，利用者ごとに異なる識別情報を Java プログラムに埋め込む電子透かし手法を提案した．

著作権所有者の識別情報を電子透かしとして埋め込む場合は，各利用者に配布されるプログラムは同一である．一方，各利用者の識別情報を電子透かしとして埋め込む場合は，電子透かしが埋め込まれる部分が異なっている．このため，複数の利用者が結託し，ソースコードから異なっている部分を探し出すことにより，電子透かしの位置が簡単に特定される可能性があった．しかし，クラス構造変換手法を適用することで，利用者ごとに異なる位置に電子透かしを埋め込むことが可能となり，電子透かしの埋め込み位置の特定を難しくすることができた．

参 考 文 献

1) Berghel, H.: Watermarking cyberspace, *Comm. ACM*, Vol.11, No.40, pp.19–24 (1997).
2) Chan, J.-T. and Yang, W.: Advanced obfuscation techniques for Java bytecode, *Journal of Systems and Softwarem*, Vol.71, No.1–2, pp.1–10 (2004).
3) Collberg, C.: SandMark: A Tool for the Study of Software Protection Algorithms. http://cgi.cs.arizona.edu/~sandmark/sandmark.html
4) Collberg, C. and Thomborson, C.: Software Watermarking: Models and Dynamic Embeddings, *Proc. Symposium on Principles of Programming Languages, POPL'99*, pp.311–324 (1999).
5) Collberg, C.S., Huntwork, A., Carter, E. and Townsend, G.M.: Graph Theoretic Software Watermarks: Implementation, Analysis, and Attacks, *Proc. Information Hiding, 6th International Workshop, IH 2004*, Lecture Notes in Computer Science, Vol.3200, pp.192–207 (2004).
6) Craver, S., Memon, N., Yeo, B. and Yeung, M.M.: Resolving rightful ownerships with invisible watermarking techniques, *IEEE Journal on Selected Areas in Communications*, Vol.4, No.16, pp.573–586 (1998).
7) Fukushima, K. and Sakurai, K.: A Software Fingerprinting Scheme for Java Using Classfiles Obfuscation, *Proc. Information Security Applications, 4th International Workshop, WISA 2003*, Lecture Notes in Computer Science, Vol.2908, pp.303–316 (2003).

8) Monden, A.: jmark: A lightweight tool for watermarking Java class filess (2002). http://se.aist-nara.ac.jp/jmark/
9) Sosonkin, G.N.M. and Memon, N.: Software and systems: Obfuscation of design intent in object-oriented applications, *Proc. ACM Workshop on Digital Right Management* (2003).
10) Sakabe, Y., Soshi, M. and Miyaji, A.: Java Obfuscation with a Theoretical Basis for Building Secure Mobile Agents, *Proc. 7th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security, CMS 2003*, Lecture Notes in Computer Science, Vol.2828, pp.89–103 (2003).
11) Thomborson, C., Nagra, J., Somaraju, R. and He, C.: Tamper-proofing software watermarks, *Proc. 2nd workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*, pp.27–36, Australian Computer Society, Inc. (2004).
12) Venkatesan, R., Vazirani, V. and Sinha, S.: A Graph Theoretic Approach to Software Watermarking, *IHW '01: Proc. 4th International Workshop on Information Hiding*, pp.157–168, Springer-Verlag (2001).
13) 門田暁人，松本健一，飯田　元，井上克郎，鳥居宏次：Java クラスファイルに対する電子透かし法，情報処理学会論文誌，Vol.41，No.11，pp.3001–3009 (2000).
14) 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）予稿集 (1998).

(平成 16 年 11 月 26 日受付)
(平成 17 年 6 月 9 日採録)

福島　和英（正会員）

2002 年九州大学工学部電気情報工学科を飛び級のため中退．2004 年同大学大学院システム情報科学府修士課程修了．同年（株）KDDI 研究所入社．ソフトウェアの著作権保護技術，暗号プロトコルの研究に従事．情報処理学会コンピュータセキュリティシンポジウム 2002 年学生論文賞，2004 年論文賞受賞．電子情報通信学会会員．

田端　利宏（正会員）

1998 年九州大学工学部情報工学科卒業．2000 年同大学大学院システム情報科学研究科修士課程修了．2002 年同大学院システム情報科学府博士後期課程修了．2001 年日本学術振興会特別研究員．2002 年九州大学大学院システム情報科学研究院助手．2005 年から岡山大学大学院自然科学研究科助教授．博士（工学）．オペレーティングシステム，コンピュータセキュリティに興味を持つ．電子情報通信学会，ACM 各会員．

田中　俊昭（正会員）

1986 年大阪大学大学院工学研究科通信工学専攻前期博士課程修了．同年 KDD（株）入社．現在，（株）KDDI 研究所セキュリティグループリーダー．暗号プロトコル，著作権保護，モバイルセキュリティ，次世代 IDS の研究に従事．電子情報通信学会会員．

櫻井　幸一（正会員）

1988 年九州大学大学院工学研究科応用物理専攻修士課程修了．同年三菱電機（株）入社．現在，九州大学大学院システム情報科学研究院情報工学部門教授．1997 年 9 月より 1 年間コロンビア大学計算機科学科客員研究員．2004 年 4 月より九州システム情報技術研究所第 2 研究室室長併任．暗号理論・情報セキュリティ・社会情報工学の研究に従事．博士（工学）．2000 年情報処理学会坂井特別記念賞受賞．2000 年・2004 年情報処理学会論文賞．2005 年 IPA 賞受賞．電子情報通信学会，日本数学会，ACM，IEEE 各会員．