プログラムに対する電子透かし埋め込み法の評価

指導教官：市川周一　　学籍番号：013740 八反田一宏

1　はじめに

電子透かしとは，データの中に情報を埋め込む手法である．例えばデータに著作権情報を埋め込むことで，データの不正コピーに対して著作権を主張することができる．電子署名も電子透かしと類似の概念である (本研究の範囲内では同一視して良い)．

Davidson ら [1] は，プログラムコードの基本ブロックを並べ替えて電子署名を構成する方法を考案した．基本ブロックとは，連続した命令列からなり，先頭の命令に制御が与えられ，途中で分岐することなく終端で制御を離れるものをいう [3]．実行順序さえ保持すれば，基本ブロックのアドレスは自由に変更することができる．ただし，実行順序を保持するにはジャンプ命令を挿入しなければならないので，プログラムの性能は低下する．

本研究では，Davidson らの方法を MIPS 命令セットを用いて実装し，署名前後の実行命令数比とプログラムサイズ比，埋め込み可能な情報量を定量的に評価した．

2　手法

透かしの埋め込みおよび復元方法を図 1 に示す．まず，原型となる命令列 (オリジナル順序 (a)) から，各基本ブロックのハッシュ値 (MD5) を取得する．ハッシュ値に従ってブロックを昇順にソートし (b)，各ブロックに番号を付ける (初期順序 (c))．

一般に n 個の要素を並べる方法は $n!$ 通りある．従って，命令列中の基本ブロック数を n としたとき，基本ブロックの並べ方を 1 つ選ぶことによって $\lfloor \log_2 n! \rfloor$ bit の情報を表現できる．埋め込む値に応じた順序 (署名値順列 (d)) を選択し，その順序に基本ブロックを並べ替える (署名つき順序 (e))．このとき実行順序が保たれるようにジャンプ命令を挿入する．(d) の署名値順列は，n 個の要素で構成される順列を辞書式ソートして $0, ..., (n! - 1)$ の番号をつけたものである．署名付き順序から署名値を復元するには，署名付き順序の命令列から初期順序を生成し (f)，2 つの順序を比較して署名値を算出する．

$n!$ の値は $O(n^n)$ であるから，n に従って急速に増大する．従って基本ブロック数 n が多いと，計算機上で実現することが困難になる．そこで，n 個のブロックを k 個 ($k < n$) ごとのグループに分割し，各グループごとに $\lfloor \log_2 k! \rfloor$ bit のデータを埋め込む手法を用いる (PPS; Partial Permutation Scheme [2])．PPS 採用時の埋め込み量は，$\lfloor n/k \rfloor \lfloor \log_2 k! \rfloor$ bit である．

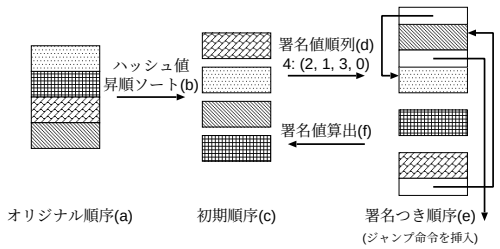

図 1: 埋め込みと復元の方法

3　実験

本研究では $k = 6$ の PPS を採用して，各グループで $\lfloor \log_2 6! \rfloor = 9$ bit の情報を表現する．9 bit のうち，8 bit をデータ，1 bit をデータ確認用の odd parity に用いる．基本ブロック数 n に対して，$\lfloor n/6 \rfloor$ byte の署名データをプログラム中に埋め込むことができる．より多くの情報を埋め込むには，k の値を大きくとればよい．

表 1 に示すプログラムに対して，ランダムに生成した 1000 個の署名データを埋め込み，プログラムサイズと実行命令数を測定した．MIPS 用クロス開発環境として，コンパイラに `egcs-2.91.66`，シミュレータに `gdb 5.2.1` を採用した．

プログラム名	行数	関数数	基本ブロック [個数]	サイズ [byte]	内容
`sort.c`	30	1	13	440	バブルソート
`findmaxmin.c`	51	1	21	576	最大・最小値検索
`optmult.c`	36	1	22	808	行列積計算
`linpackc.c`	907	12	352	20816	LINPACK ベンチマーク
`whetstone.c`	433	4	113	6280	Whetstone ベンチマーク
`dhry_1.c`	385	6	129	4612	Dhrystone Pack.1
`dhry_2.c`	192	6	52	1564	Dhrystone Pack.2
`livermore.c`	1435	1	408	34352	Livermore ベンチマーク

表 1: 実験用プログラムの情報

4　結果

表 1 から比較的小規模なプログラム 3 つ (`sort.c`, `findmaxmin.c`, `opmult.c`) を選んで実行し，シミュレータで命令数を測定した．結果を表 2 に示す．埋め込み後の実行命令数は 6〜11%増加した．

プログラム名	実行命令数		比
	埋込前	埋込後平均	
`sort.c`	1983	2140.3	1.08
`findmaxmin.c`	239	266.1	1.11
`optmult.c`	1270	1350.5	1.06

表 2: 実行命令数比

埋め込み前後のプログラムサイズ比を図 2 に示す．横軸には各プログラムの平均基本ブロック長 (命令数) を取った．基本ブロックが長いプログラムほど，プログラムサイズ比が 1.0 に近いことがわかる．

埋め込み可能情報量とプログラムサイズの比 (情報密度) を図 3 に示す．本研究の実装 ($k = 6$ の PPS) では，埋め込めるデータ量はプログラムサイズの約 0.2%となっている．

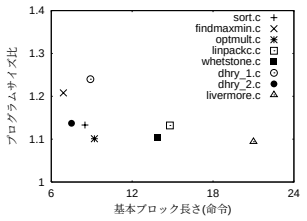

図 2: プログラムサイズ比

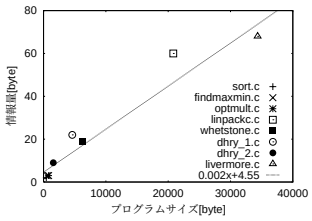

図 3: 情報密度

5　おわりに

今回は時間の関係上，実行時間の増加に関しては詳しく分析できなかった．ベンチマークプログラムを実際の MIPS プロセッサ上で実行し，署名前と署名後の性能を測定することで，より現実的なオーバヘッドが測定できると思われる．

Davidson らの方法 [1] 以外にも，透かしを埋め込む方法は考えられる．それらに関しても評価を進めて，結果を比較してみたい．

参考文献

[1] R.L.Davidson et al.: Method and system for generating and auditing a signature for a computer program, US Patent 5559884 (1996).

[2] S.Ichikawa, T.Chiyama, K.Akabane: Redundancy in 3D polygon models and its application to digital signature, Journal of WSCG, Vol.10, No.1, pp.225–232 (2002).

[3] A.V. エイホ, R. セシィ, J.D. ウルマン:コンパイラ II – 原理・技法・ツール –, サイエンス社 (1990).