IEICE TRANS. FUNDAMENTALS, VOL.E87–A, NO.1 JANUARY 2004

LETTER *Special Section on Cryptography and Information Security*

The Evaluation of Davidson's Digital Signature Scheme

Kazuhiro HATTANDA[†], ***Nonmember and*** **Shuichi ICHIKAWA**[†a)], ***Member***

SUMMARY Davidson's scheme utilizes the order of basic blocks to embed a digital signature in a computer program. To preserve the function of the original program, additional jump instructions are inserted. This involves some overhead in both size and performance. In our implementation, the increase in size was between 9% and 24%. The performance of benchmark programs was 86–102% of the original.

key words: *digital signature, watermarking, computer program, basic block, permutation*

1. Introduction

Davidson and Myhrvold [1] invented a method to generate a signature on a computer program. They pointed out that the basic blocks of a computer program can be reordered arbitrarily without changing the behavior of the program, if *jump* instructions are correctly inserted and maintained. Their idea is to embed a signature as a reordered sequence of basic blocks. The obvious drawbacks of this method are the increase in program size and the performance degradation caused by additional jump instructions.

In this paper, Davidson's scheme is examined in a quantitative manner. Though there have been many studies of digital signatures for computer programs, the authors have never come across an empirical evaluation of Davidson's scheme. Thus, this study would be a worthwhile step for future studies.

2. Davidson's Scheme

Figure 1 illustrates the basis of Davidson's scheme. First, the hash values are calculated for each basic block in the original code (a). In this study, MD5 is adopted as the hash function. If necessary, we can add NOP instructions to avoid hash collisions. Second, blocks are sorted according to their hash values (b) to normalize the order of basic blocks (c). Third, we reorder basic blocks arbitrarily to embed a signature (d). To preserve the function of the original code, we must add some jump instructions wherever necessary (e). We can decode a signature from the order of basic blocks (f), by comparing the signed order (e) to the initial order (c).

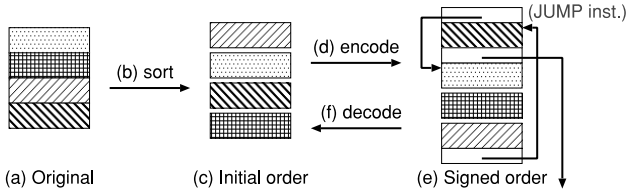

Fig. 1 Davidson's scheme.

Generally speaking, we have $n!$ options in ordering n items. That is, we can embed $\lfloor \log_2 n! \rfloor$ bits of information in n items. It is a problem that $n!$ is $O(n^n)$ and difficult to handle when n is large.

It is natural and easy to split n items into groups. Let k be a small constant ($k \leq n$). Each k items make a group. As each group can carry $\lfloor \log_2 k! \rfloor$ bits, the total capacity is given by $\lfloor n/k \rfloor \cdot \lfloor \log_2 k! \rfloor$ bits. This scheme (Partial Permutation Scheme; PPS) was mentioned by Ichikawa, Chiyama, and Akabane [2] for 3D polygon models.

In this study, the PPS of $k = 6$ is implemented. Though each group can carry $\lfloor \log_2 6! \rfloor = 9$ bits, only 8 bits are used for the signature and one bit is used for a parity bit to check integrity. Thus, the total capacity of n basic blocks is $\lfloor n/6 \rfloor$ bytes. For denser embedding, we have to choose a larger k.

3. Program Size and Watermark Capacity

First, the increase in program size is investigated. Table 1 summarizes 8 program files, which include famous benchmark programs `linpackc`, `whetstone`, `dhry_1`, `dhry_2`, and `livermore`. These benchmark programs were downloaded from Netlib [3]. The target architecture is MIPS, which is simple and easy to handle. All evaluations in this section were performed with MIPS cross-development tools which include egcs-2.91.66 and gdb-5.2.1.

The column "Original" in Table 1 lists the sizes of the original object codes. The column "Signed" lists the average size of 100 signed codes. We generated 100 random signatures for each program, embedded each of them in the object code, and then measured the size of each signed code. The increase in size was 9–24% of the original size.

Manuscript received March 24, 2003.
Final manuscript received August 19, 2003.

[†]The authors are with the Department of Knowledge-based Information Engineering, Toyohashi University of Technology, Toyohashi-shi, 441-8580 Japan.
a) E-mail: ichikawa@tutkie.tut.ac.jp

Table 1 Program statistics.

Name	Note (Compile Option)	#Line	#Func	#Basic Block	Original [byte]	Signed [byte]	Watermark [byte]
`sort.c`	Bubble sort (`-O0`)	30	1	13	440	498.4	2
`findmaxmin.c`	Search Max and Min (`-O0`)	51	1	21	576	696.6	3
`optmult.c`	Matrix Multiplication (`-O0`)	36	1	22	808	888.8	3
`linpackc.c`	LINPACK (`-O0 -DDP -DUNROLL`)	907	12	378	23460	26409.0	61
`whetstone.c`	Whetstone benchmark (`-O0`)	433	4	113	6280	6835.3	18
`dhry_1.c`	Dhrystone (`-O0 -DHZ=100 -DTIME`)	385	6	129	4612	5718.0	20
`dhry_2.c`	Dhrystone (`-O0`)	192	6	52	1564	1779.2	7
`livermore.c`	Livermore benchmark (`-O0`)	1435	1	408	34352	37595.3	67

Table 2 Benchmark results.

Program	#loop	Original (100 trials)			Signed (100 random signatures)		
		Maximum	Average	Minimum	Maximum	Average	Minimum
`dhry` [Dhrystone]	`10000000`	212766.0	211170.2	208333.3	192307.7	186981.4	181818.2
`linpackc` [Kflops]	`NTIMES=10`	23100.0	23078.6	22978.0	22616.0	22261.4	21890.0
`whetstone` [MIPS]	`100000`	123.5	122.2	120.5	125.0	121.6	120.5

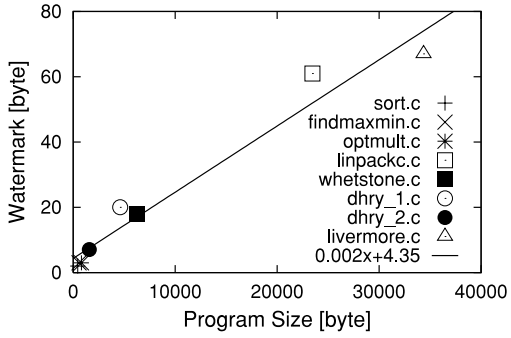

Fig. 2 Watermark size.

As stated earlier, watermark capacity is bound by $\lfloor n/6 \rfloor$. However, there are more factors that prevent us from utilizing basic blocks. For example, the first block of each function is excluded from permutation to keep the entry address. The "Watermark" column in Table 1 lists the watermark capacity that is actually available after all these factors.

Figure 2 displays the relationship between code size and watermark capacity. Though watermark capacity is not directly related to code size, Fig. 2 shows that they correlate to some extent. From the regression line by least-squares method, the watermark capacity is approximated by the 0.2% of code size in the case of PPS ($k = 6$).

4. Performance Overhead

In this section, the performance overhead is examined on a real MIPS platform. SGI Onyx Infinite Reality is powered by MIPS R10000 (195 MHz) processor with 1GB main memory, IRIX 6.5.5 m, and gcc-3.0.1. Though Onyx is a parallel computer with 4 processors, we only used one processor for performance evaluation.

Table 2 summarizes the results of Dhrystone, Linpack, and Whetstone benchmarks. First, the original benchmarks were measured 100 times, and their maximum/average/minimum values are reported. The "Signed" column shows the average performance of 100 signed codes for 100 random signatures. The performance degradation was modest (0–11%) for these benchmark programs.

It is very interesting that the best performance of signed whetstone codes is 2% better than the average of the original code. Moreover, it is even better than the best result of the original code. There is no wonder here because (1) redundant jump instructions can be removed, and (2) the reordered code can show more locality than the original. In fact, Davidson et al. [4] have already patented this technique.

5. Conclusion

There are many other watermarking schemes for computer programs. More extensive surveys and quantitative investigations must be undertaken.

Acknowledgment

This work was partially supported by a grant from the Telecommunications Advancement Foundation (TAF).

References

[1] R.I. Davidson and N. Myhrvold, "Method and system for generating and auditing a signature for a computer program," U.S. Patent 5,559,884, Sept. 1996.
[2] S. Ichikawa, H. Chiyama, and K. Akabane, "Redundancy in 3D polygon models and its application to digital signature," J. WSCG, vol.10, no.1, pp.225–232, 2002.
[3] Netlib Repository. http://www.netlib.org/
[4] R.I. Davidson, et al., "Method and system for improving the locality of memory references during execution of a computer program," U.S. Patent 5,664,191, Sept. 1997.