Software Copy Detection Based on Watermark in Output Content

Kazuhide Fukushima, Shinsaku Kiyomoto, and Yutaka Miyake
KDDI R&D Laboratories, Inc.
2–1–15 Ohara, Fujimino, Saitama, 356–8502, JAPAN
Email: {ka-fukushima,kiyomoto,miyake}@kddilabs.jp

***Abstract*—This paper proposes a copy detection framework for software based on watermarks embedded in output. Our framework targets enterprise software that generates commercial content where the number of users is limited. We provide four candidates for embedded information and compare them in terms of security and efficiency. Then, we propose watermarking mechanisms for programs and VHDL code in order to apply our framework to detecting the copying of software development kits and hardware design tools. Experimental results show that we can securely embed watermarks in VHDL code with a feasible overhead.**

I. Introduction

The worldwide economic loss caused by software piracy was estimated to be 59 billion dollars in 2010 [1]. Copy protection schemes have been proposed to deal with this problem, but many of these schemes have been cracked due to the limitations of software-based approaches. In online license checking schemes, software accesses a license management server to register ownership of a user when the software is installed or executed. These schemes have the drawback that the software vendor must manage a license management server, and the software requires Internet access. Offline checking schemes eliminate these problems. Generally, these schemes use a tamper-proof device in order to manage license information and/or a license-checking mechanism. However, they impose additional cost and effort on users. Another solution is to embed information in the output as a watermark. This solution is not a real-time solution, but the software execution does not require online access or special hardware.

In this paper, we propose a copy detection framework based on watermarks embedded in output content. We present four kinds of information to be embedded as a watermark: 1) software ID associated with a user, 2) software ID and PC information, 3) PC information and execution dates, 4) software ID, PC information and execution dates. Then, the transformation process for the information and watermarking mechanisms for programs are discussed. Next, we propose watermarking mechanisms based on a software obfuscation technique and apply our framework to the detection of copied hardware design tools. We evaluate the size of the watermarked VHDL code for an electronic circuit in order to show the applicability of our watermarking mechanism.

The rest of the paper is organized as follows: related work is summarized in Section II. Section III provides our copy detection framework based on watermarks embedded in output content. We compare the candidates for embedded information and watermarking mechanisms in terms of security and efficiency. We apply our framework to the protection of software development kits and hardware design tools in Section IV. Watermarking mechanisms based on a software obfuscation technique for programs and VHDL code are proposed. We conclude this paper in Section V.

II. Related Work

Digital watermarking is a technique that embeds auxiliary information in content. This information is used to prove the ownership of copyright of content or to trace a user who distributes content. Previously, digital watermarking schemes aimed mainly at directly copyrighted content such as image data, audio data, and text data. These kinds of content have a high level of redundancy, so in many cases, a user cannot recognize that the content is different even when several of its bits have been changed. Watermarking schemes for software have also been proposed in order to prove unauthorized copying of software. Monden, et al. [2] proposed a scheme for Java byte codes that embeds a watermark in opcodes and numeric operands in dummy methods. Venkatesan et al. [3] proposed a scheme that turns a watermark into a control flow graph and merges it into the graph of the original software. These are static watermarking schemes, that is, the watermark can be detected without executing software. Collberg et al. [4] and Thomborson et al. [5] proposed a dynamic watermarking scheme, in which a watermark is output when the software is executed with specific input. After that, several dynamic schemes [6], [7] have been proposed. Watermarking schemes for hardware description languages have also been proposed. Yuan et al. [8] proposed schemes that embed watermarks in redundant hardware descriptions for *don't-care* conditions.

A software obfuscation scheme transforms original source code or a binary program into an obfuscated source code or program that is more complicated and difficult to analyze, while still preserving its functionality. Barak et al. [9] showed the existence of classes of functions that

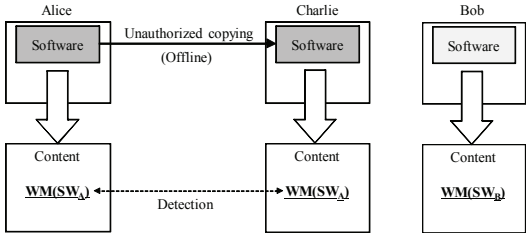

Fig. 1. Proposed framework

cannot be obfuscated. However, some practical obfuscation schemes have been proposed. These schemes are intended to make the cost of analyzing a program higher than the value of the program. Gannod et al. [10] proposed an obfuscation scheme that obfuscates loops. Collberg et al. [11] proposed a scheme that inserts dummy instructions using conditional branching. Chan et al. [12] proposed a scheme that modifies identifiers contained in Java byte code in order to protect the code against decompilation. Sosonkin et al. [13] proposed a scheme that changes the structure of classes in a Java code by merging and dividing them. Obfuscation schemes with a theoretical basis have also been proposed. Wang et al. [14] and Sakabe et al. [15] proposed obfuscation schemes based on NP-hard problems. Collberg et al. [16] and Fukushima et al. [17] have both proposed obfuscation schemes that encode variables. There are also some software obfuscation schemes that target VHDL code [18], [19], [20].

III. Copy Detection Framework

We propose a framework that enables software vendors to detect unauthorized software copying by discovering watermarks embedded in output content. The target software automatically embeds a watermark into output.

Figure 1 shows our framework. The software vendor can detect unauthorized copying of the software by extracting watermarks $WM(SW_A)$ from the output content. The detection process varies according to information that embedded as a watermark; the detailed process is described in Section III.B. We do not always need to determine that Alice gave a copy of the software to Bob. Instead, it is important to detect the fact of copying. There are many existing studies on watermarking schemes to confirm the distribution or reuse of content. Thus, we discuss only an embedding mechanism for $WM(SW)$ in the following sections of this paper.

We will show the assumptions in Section III.A. Section III.B discusses the embedded information and detection process. Information transformation and watermarking mechanisms are described in Section III.C and Section III.D, respectively.

A. Assumptions

Our framework targets enterprise software that generates commercial content for which the number of users is limited. In this situation, detection of unauthorized copying of the software is important, as well as identification of malicious users. One effective approach is to warn all the users that the software vendor will take legal action. The threat of legal action causes a user to hesitate to distribute the software on the Internet. Instead, the user may give a copy of the software to a colleague offline. The colleague uses the software in order to distribute or sell content. Our framework thus focuses on a watermark embedded in output content.

We assume that a secure software obfuscation scheme for the software is available. In our framework, a watermark embedding function added to the target software is essential. If this function is removed or bypassed, the vendor can neither identify illegal users nor detect the fact of unauthorized copies; that is a limitation of our framework. Thus, we have to obfuscate the software in order to protect the watermark embedding function against analysis, alteration, or removal.

B. Embedded Information

We consider four candidates for embedding as a watermark: 1) software ID associated with a user, 2) software ID and PC information, 3) PC information and execution dates, and 4) Software ID, PC information and execution dates.

1) Software ID Associated with a User: The software embeds the software ID in its output. We assume that software ID is associated with a user when the software is purchased. A record of the software ID assigned to each user name must be kept by the software vendor. Copying of the software is detected by a watermark indicating that the provider of some content is different from the registered user of the software.

Assume that the software has the software ID SW_A associated with Alice. She gives a copy of the software to Bob. Then, Bob executes the software and obtains content. The software vendor can detect the unauthorized copying from the watermark extracted from Bob's content. The watermark indicates that the original user of the software was Alice.

2) Software ID and PC Information: The software embeds both the software ID and PC information in its output. Unauthorized copying of the software is detected by finding watermarks that indicate that the same software was executed on two or more PCs.

We assume Alice has the software SW_A. She executed the software on her PC PC_A and generated content. Then, she gave a copy of the software to Bob. Bob executed the software on his PC PC_B and generated other content. The software vendor can detect the copying of the software from the watermarks $\langle SW_A, PC_A\rangle$ and $\langle SW_A, PC_B\rangle$. These watermarks indicate SW_A was executed on PC_A and PC_B.

3) PC Information and Execution Dates: The software embeds both PC information and execution dates in the

output content. This approach requires a trusted clock as well as a timestamp technique [21]. Software copying is detected by finding a watermark that indicates the software was executed on two or PCs.

Assume that Alice executes the software at 9:00 on January 1st and at 9:00 on Jan 2nd on PC PC_A, and gives a copy of the software to Bob. Then, Bob executes the software at 9:00 on January 3rd on PC PC_B. The vendor can detect the copying from the watermark extracted from Bob's content. The watermark indicates that the software has been executed on two PCs.

4) Software ID, PC Information, and Execution Dates: The software embeds the software ID, PC information and execution dates in the output content. Software copying is detected by finding the telltale watermarks, as in the two cases above. The vendor can detect software copying from watermarks that have the same software ID together with inconsistent execution dates.

Assume that Alice executes the software SW_A at 9:00 on January 1st and at 9:00 on January 5. Bob executes the copied software SW_A at 9:00 on January 3rd. The software vendor can detect the copying from the watermarks. The two watermarks contain the same software ID, but the execution dates are inconsistent.

5) Comparison: We compare the four candidates for embedded information in terms of security, usability, efficiency of copy detection, and watermark size. Table I summarizes our comparison.

a) Security: Software copying within the same office is difficult to detect when only the software ID is embedded in the output content. A user gives a copy of the software to a colleague. The colleague generates content using the software and sells the content. In this situation, the colleague can claim that the content was generated by the original user.

A user can avoid detection by conducting a back-up attack if the execution dates are embedded. The user can make a copy of the software with no execution dates by backing it up. The back-up attack is possible since an execution time is dynamic data and different for every execution. However, software ID and PC information are static data and secure against this attack.

b) Usability: Embedding the PC information implies a strong restriction, i.e., a user must work on one unique machine with the same components. This issue can be solved by notifying the change of the target PC to the vendor. When a user wants to replace some parts of PC or entire PC, one needs to notify the new PC information to the vendor. The vendor replaces the old PC information with the new one in the detection process.

c) Efficiency of Copy Detection: Unauthorized copying can be detected from only one piece of content if the software ID associated with a user, or execution dates are embedded. However, we need two or more pieces of content to detect copying if both the software ID and PC information, or both the software ID and execution dates are embedded. We have to find two watermarks that contain the same software ID and different PC information.

d) Watermark Size: Software ID with 16 bits is sufficient, since the number of users is limited. We need an identifier with at least 32 bits to identify the hardware information for a PC, considering the overall number of PCs. We can use hardware information such as the identifier of a processor, hard disk drive, motherboard or the MAC address of a network interface. However, the MAC addresses of some network interface devices can be altered. Furthermore, a user may replace a processor or a hard disk drive. These identifiers may not uniquely specify a PC. We may use the identifier of a motherboard to generate PC information, since the motherboard is considered to be the most essential component of a PC. Operating systems such as UNIX and Microsoft Windows use 32 bits to identify the date and time, and this can be used as the execution time. The datum is the number of seconds from January 1st 1970, or from 1900, to now. We have to embed a watermark with $32e$ bits where e is the number of executions.

C. Information Transformation

In the first step of embedding information, the software transforms the information into secure encrypted data. First, the software calculates the hash value of the information and concatenates the hash value with the information, and then the software encrypts the data using the software vendor's key. A public-key based random encryption algorithm such as AES-WRAP [22] or RSA-OAEP [23], [24] is used for the encryption to detect data corruption. The key is securely embedded in valid software by the software vendor. The vendor has a private key for decrypting the data. The vendor obtains encrypted data from content, decrypts it, and verifies the hash value of the information. The objective of the transformation is as follows:

- Alteration of the information can be detected. If an attacker without the key alters watermarked content, the vendor can detect the alteration. The vendor cannot correctly decrypt the information or fails to verify the information.
- Embedded information becomes pseudo-random data. The software transforms the information into pseudo-random data using an encryption algorithm. Different embedded data are generated for each transformation from the same information. With randomized embedded data, the attacker will find it more difficult to discover the hiding techniques.
- Embedded information becomes protected data. A valid software vendor who has the private key can decrypt the information and detect unauthorized copying. The information may include sensitive information related to a user's privacy. Thus, the information should only be traceable by the vendor.

TABLE I
COMPARISON OF WATERMARKING CANDIDATES

Information	Assumptions	Advantages	Disadvantages
Software ID	User registration	Fixed-size watermark Copy detection from one output Secure against backup copying	Local copying
Software ID and PC information	Unique PC info.	Fixed-size watermark Secure against backup copying	Copy detection from multiple outputs
PC information and execution dates	Trusted clock Unique PC information	Copy detection from one output	Backup copying Variable-size watermark
Software ID, PC information, and exec. dates	Trusted clock Unique PC information	Copy detection from one output Multiple detection	Backup copying Variable-size watermark

D. Watermarking Mechanisms

Existing digital watermarking schemes [25], [26] are available for content such as image data, audio data, and text data. On the other hand, some enterprise software (e.g. software development kits and hardware design tools) outputs a program or source code as output content. We have proposed watermarking mechanisms based on a software obfuscation technique that can be applied to programs and source code written by a hardware description language.

IV. Case Study

We apply the proposed framework for detecting the copying of software development kits and hardware description tools. A software development kit is a group of tools to develop applications. Various software development kits for application frameworks, operating systems and embedded systems are available. However, hardware design tools are used to design electronic circuits such as LSI systems. They enable electronic circuits be designed using a GUI interface, and they output source codes in a hardware description language. Enterprise software development kits and hardware design tools are expensive, and unauthorized copying seriously affects the revenue of vendors. Many copy protection schemes have been proposed, but there is no perfect solution. Thus, we need not only techniques to prevent unauthorized copying but also a framework for detection.

We apply our proposed framework to protect these kinds of software. The output is a program or source code written by a hardware description language. Generally, programs and source code have less redundancy than normal digital content. It is difficult to efficiently embed a watermark in them. We first propose watermark embedding mechanisms for programs in Section IV.A. These mechanisms can be applied to the source code of VHSIC Hardware Description Language (VHDL). We evaluate the size of the watermarked VHDL code in Section IV.B. Section IV.C and IV.D evaluate the security and efficiency of these watermarking mechanisms.

A. Application to Software Development Kits

The output of a software development kit is a program. We thus propose watermarking mechanisms for programs. The proposed mechanisms use a software obfuscation technique: the xor-encoding scheme [17]. The obfuscation scheme encodes multiple variables simultaneously using a Boolean matrix and an integer vector. The encoding process uses only exclusive-or operations.

1) Mechanisms Using Dummy Instructions: Monden et al. [2] proposed a watermarking scheme for a Java program that embeds a watermark using a dummy method. We can add dummy instructions that are never actually executed. Thus, we can keep the functionality of the software. A watermark can be embedded in numerical data and operations in these dummy instructions. The operation + can be replaced with `-`, `*`, `/`, `%`, `and`, `or`, or `xor`. One of eight operations can be replaced with another without violating the grammar[1]. We can embed 3-bit information by replacing this information with any of these eight opcodes when we encounter one of them in a program. That is, 000 is assigned to `+`, 001 to `-`, 010 to `*`, ..., and 111 to `^`.

In this mechanism, a sophisticated predicate is required to conceal the fact that the instructions in the `if` statement are dummy instructions. If we use a trivial predicate such as `i xor i < 0`, a user may realize that these instructions are dummy instructions for embedding a watermark. The user may remove or alter these instructions to modify the watermark. We need a predicate that is difficult to analyze, but is always evaluated as *false*. This predicate is called an opaque predicate. We can construct an opaque predicate using the xor-encoding scheme. This obfuscation scheme encodes multiple variables using a matrix and vector as follows:

$$\begin{pmatrix} \mathtt{A} \\ \mathtt{B} \\ \mathtt{C} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathtt{a} \\ \mathtt{b} \end{pmatrix} \oplus \begin{pmatrix} 10010110 \\ 01100101 \\ 10100101 \end{pmatrix} \quad (1)$$

In this situation, two variables `a` and `b` are encoded to `A`, `B`, and `C`. Encoded variables `A`, `B`, and `C` satisfy the non-trivial relation $\mathtt{A} \oplus \mathtt{B} \oplus \mathtt{C} \oplus 01010110 = 0$. Thus, we can construct an opaque predicate using this relation. For example, `if ((A^B^C^01010110) < 0)` is a false opaque predicate. On the other hand, normal predicate `if (a < 0)`, which is

[1]Note that the replacements change the semantics of the code. However, these instructions are never executed and do not affect the original code.

TABLE II
Evaluation environment

FPGA designing tool	Xilinx ISE 10.1
Synthesis tool	Xilinx Synthesis Tool
Simulator	Modelsim-XE VHDL
Generated simulation language	VHDL

TABLE III
Watermarking mechanism with opaque predicate [27]

Watermark size [bits]	8	16	32	64	128
Number of slices	8	8	8	13	15
Number of flip-flops	8	8	8	8	8
Number of LUTs	15	17	17	30	26

TABLE IV
Watermarking mechanism using our proposed opaque predicate

Watermark size [bits]	8	16	32	64	128
Number of slices	8	11	13	15	20
Number of flip-flops	14	12	16	16	28
Number of LUTs	20	22	27	29	42

TABLE V
Watermarking mechanism using software obfuscation

Watermark size [bits]	8	16	32	64	128
Number of slices	14	22	36	36	40
Number of flip-flops	14	12	28	35	38
Number of LUTs	20	22	47	69	76

sometimes evaluated as *true* and sometimes as *false*, is transformed into `if (A^B^11110011 < 0)`. It is difficult to distinguish between an opaque predicate and a normal predicate since they have the same form.

2) Mechanism Using Software Obfuscation Technique: We propose watermark embedding mechanisms based on the above obfuscation scheme. We can embed a watermark in a dummy variable by assigning arbitrary data to this variable. However, an attacker can specify the dummy variable since the value of this variable is not used. To address this problem, we use the xor-encoding scheme. This obfuscation scheme makes it difficult to specify the dummy variable since the value of the dummy variable is mixed with those of the existing variables. Only the software vendor, who knows the key, can decode the variables and obtain the watermark.

B. Application to Hardware Design Tools

The output of a hardware design tool is source code written for hardware description language. Thus, we select VHDL code as our watermarking target. The embedded information and watermarking mechanisms are evaluated by comparing FPGA circuit descriptions that include watermarks. The circuit size is examined with respect to the following quantities:

- The number of slices which is a functional unit on an FPGA circuit. Their number is determined by the number of flip-flops and look-up tables (LUTs).
- The number of flip-flops.
- The number of LUTs with four inputs.

The evaluation environment is shown in Table II. We show the watermarked VHDL code in the Appendix.

We made a VHDL code for an 8-bit counter as a sample code. Figure 2 shows the code. The circuit generated from the original VHDL code has 8 slices, 8 flip-flops, and 15 LUTs. To begin with, we embedded a watermark by using the watermarking mechanism based on dummy instructions. This mechanism relies on opaque predicates. We used the following opaque predicates: a numerical predicate proposed by Collberg et al. [27] (lecture 13, section H, (1)) based on encoded variables. We made five circuits with 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit watermarks. Table III shows the size of the circuits generated from watermarked VHDL codes. Then, we used a false opaque predicate `if(A xor B xor C xor 01010110 < 0)` based on Equation (1), and made five circuits with watermarks. Table IV shows the size of the circuits with watermarks. These tables show that the watermaking mechanism with the obfuscation technique requires larger overhead size except for a case. The case seems to beattributed to an inefficientlogic synthesis by the tool. Finally, we embedded watermarks in VHDL codes using the obfuscation technique, and obtained circuits. We introduced redundant variables to reserve the space for embedding watermarks. For example, an assignment instruction `a = b;` can be transformed to `c = b; a = c;` where `c` is a redundant variable. Table V shows the circuit size. Compilers or logic s tools may remove the redundant code. However, it is difficult for these tools to the redundant variables of an obfuscated code. Note that all the variables are used to calculate the next value in the obfuscated code.

We can apply the proposed framework while keeping a reasonable circuit size overhead by embedding either the software ID or both the software ID and PC information. The execution dates are not suitable to be embedded in programs and VHDL code due to their large data size.

C. Security Analysis

We define three attacks on watermarking mechanisms. Then, we discuss the security of the proposed mechanism when they are applied to programs and VHDL code.

1) Attack Model: We consider the following attacks in our security evaluation.

a) Static Attack.: Static attacks analyze watermarked content without executing the program or implementing the electronic circuit. An attacker identifies where watermarks are embedded, and alters or removes them.

b) Dynamic Attack.: Dynamic attacks analyze watermarked content by executing the program or implementing the electronic circuit. An attacker can user a program debugger or logic simulator to efficiently identify the redundant parts of a program or VHDL code.

c) Coalition Attack.: Watermarked content is distinct for every user. In coalition attacks, users can identify

the position of the watermark by comparing two or more pieces of content generated by distinct software.

2) Programs: Watermarking mechanisms based on redundant descriptions and dummy instructions are not robust. If an attacker realizes that there are redundant parts in a program, the attacker can alter or remove the watermark. We can improve security against static attacks by using sophisticated opaque predicates, but they still remain vulnerable to dynamic attacks. The attacker may find dummy functions and opaque predicates by using debuggers. A watermarking mechanism using software obfuscation can protect against both static and dynamic attacks. Static attacks against the software obfuscation technique require $\Omega(2^{m^2})$ computational complexity [17] where m is the number of encoded variables. Furthermore, the dummy data is mixed with the existing data and stored in multiple variables. All of the variables are used when the program is executed. Thus, this mechanism is sufficiently robust to resist dynamic attacks.

3) VHDL Code: Watermarking mechanisms based on redundant descriptions and dummy instructions are not robust. If an attacker realizes that there are redundant parts in VHDL code for electronic circuits, the attacker can alter or remove the watermark. We can improve the security against static attacks by using sophisticated opaque predicates, but they still remain vulnerable to dynamic attacks. A watermarking mechanism based on software obfuscation can protect against both static and dynamic attacks. Static attacks against the software obfuscation technique require $\Omega(2^{m^2})$ computational complexity [17] where m is the number of encoded variables. Furthermore, dummy data is mixed with existing data and stored in multiple variables. All of the variables are used and updated in the electronic circuit implemented by the VHDL code. Thus, this mechanism is sufficiently robust to resist dynamic attacks.

We can use a software obfuscation technique to achieve security against coalition attacks. For example, there are sophisticated tools [28] to compare different versions of the same software. Obfuscation can provide a distinct transformation for each user so that straightforward comparison does not work. In this case, a desirable technique is a probabilistic scheme or scheme using a parameter that controls the transformation. Our obfuscation scheme [17] can satisfy the requirement by randomly changing the encoding rules.

D. Efficiency Analysis

We discuss the efficiency of the proposed mechanism as applied to programs and VHDL code. The efficiency is evaluated by the increase of program size or circuit size designed by the watermarked VHDL code.

1) Programs: Mechanisms based on redundant descriptions and mechanisms using dummy instructions do not affect the execution efficiency of the programs. The size of the program increases in proportion to the watermark size.

On the other hand, a mechanism based on software obfuscation increases the program size and execution time by $O(m^3)$ [17] where m is the number of encoded variables. These overheads can be reduced further by applying the obfuscation technique only to limited parts of programs.

2) VHDL Code: A mechanism based on redundant descriptions, and mechanisms using dummy instructions, increase the size of the electronic circuit designed by the watermarked VHDL code in proportion to the watermark size. However, this does not have a significant impact on the propagation delay since the dummy circuit is only used for watermark extraction and does not affect the essential parts of the original circuit.

A mechanism based on the software obfuscation technique increases the size of the electric circuit by $O(m^3)$. The increase in the propagation delay stays within $O(m)$ with a parallelization technique.

These overheads can be reduced further by applying the obfuscation technique only to limited parts of VHDL code. Table III, IV, and V show the size of an electronic circuit implemented by the watermarked content. The number of flip-flops does not increase when opaque predicates [27] are used, and the size increase stays within a factor of two for a 128-bit watermark when our proposed predicates are used. Our experimental results in Table V show that the increase in the number of flip-flops is less than tripled (from 14 to 42) for a 128-bit watermark. Watermarking mechanisms based on dummy instructions are suitable for small programs or VHDL code where the allowable overhead is limited. A mechanism based on the software obfuscation technique is suitable for complicated or valuable outputs where the allowable overhead is relatively large.

3) Limitations and Improvements: Our framework embeds a watermark into output of the target software. The users of software may consider that it may make a fatal impact on debugging and verification processes. In this case, we can use watermarking mechanisms based on opaque predicates. The watermarks is embedded into the dummy parts do not affect the original functions.

When our framework is applied to a hardware design tool, it is difficult to extract a watermark directly from a chip. Some chips have physical protections like photosensitive or air-sensitive destruction systems. In this case, we should use the dynamic watermark mechanisms [4], [5], [6], [7]. A watermarked chip returns special output as a watermark for specific input.

V. Conclusion

In this paper, we propose a software copying detection framework based on watermarks embedded in output. Our framework targets expensive software where the number of users is limited. We investigate 1) embedded information as a watermark, 2) information transformation, and 3) watermarking mechanisms for programs. Our evaluation

shows that the embedded information and watermarking mechanisms involve tradeoffs between security and efficiency. The proposed watermarking mechanisms target programs and VHDL code as outputs. Our framework can be used to detect the copying of software development kits and hardware design tools by using these mechanisms. Experimental results show that we can securely embed watermarks in a VHDL code with a feasible overhead.

Acknowledgment

The authors would like to thank anonymous reviewers and program committee members for useful comments on an earlier version of this manuscript.

References

[1] Business Software Alliance, "Eighth annual BSA global software 2010 piracy study," http://portal.bsa.org/globalpiracy2010/downloads/study_pdf/2010_BSA_Piracy_Study-Standard.pdf, 2011.

[2] A. Monden, H. Iida, K. Matsumoto, K. Inoue, and K. Torii, "A practical method for watermarking java programs," in *Proc. 24th Computer Software and Applications Conference (COMPSAC2000)*, 2000, pp. 191–197.

[3] R. Venkatesan, V. Vazirani, and S. Sinha, "A graph theoretic approach to software watermarking," in *Proc. 4th International Information Hiding Workshop (IHW2001), Lecture Notes in Computer Science 2137*, 2001, pp. 157–168.

[4] C. Collberg and C. Thomborson, "Software watermarking: Models and dynamic embeddings," in *Proc. Principles of Programming Languages 1999 (POPL1999)*, 1999, pp. 311–324.

[5] C. Thomborson, J. Nagra, R. Somaraju, and C. He, "Tamper-proofing software watermarks," in *Proc. 2nd Australasian Information Security Workshop (AISW2004)*, 2004, pp. 27–36.

[6] X. Zhang, F. He, and W. Zuo, "Hash function based software watermarking," in *Proc of Advanced Software Engineering and Its Applications (ASEA2008)*, 2008, pp. 95–98.

[7] Y. Ke-xin, Y. Ke, and Z. Jian-qi, "A robust dynamic software watermarking," in *Proc. of 2009 International Conference on Information Technology and Computer Science (ITCS2009)*, 2009, pp. 15–18.

[8] L. Yuan, P. R. Pari, and G. Qu, "Soft ip protection: Watermarking hdl codes," in *Proc. 6th International Information Hiding Workshop (IHW2004)*, 2004, pp. 224–238.

[9] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yangpages, "On the (im)possibility of software obfuscation," in *Proc. Advances in Cryptology (CRYPTO2001), Lecture Note in Computer Science, vol.2139*, 2001, pp. 1–18.

[10] G. C. Gannod and B. H. C. Cheng, "Using informal and formal techniques for the reverse engineering of c programs," in *Proc. IEEE International Conference on Software Maintenance 1996 (ICSM'96)*, 1996, pp. 265–275.

[11] C. Collberg and C. Thomborson, "Watermarking, tamperproofing, and obfuscation — tools for software protection," *IEEE Transactions on Software Maintenance*, vol. 28, no. 8, pp. 735–746, 2002.

[12] J. T. Chan and W. Yang, "Advanced obfuscation techniques for java bytecode," *Journal of Systems and Software*, vol. 71, no. 1–2, pp. 1–10, 2004.

[13] M. Sosonkin, G. Naumovich, and N. D. Memon, "Obfuscation of design intent in object-oriented applications," in *Proc. 3rd ACM workshop on Digital rights management (DRM2003)*, 2003, pp. 142–153.

[14] C. Wang, J. Hill, J. Knight, and J. Davidson, "Software tamper resistance: obfuscating staticanalysis of programs," in *Tech. rep. SC-2000-12, Dept. of Computer Science, University of Virginia*, 2000.

```
counter:process (clock);
  begin
    count <= "0";
    if (clock'event and clock = "1") then
      count <= count + 1;
    end if;
  end process;
```

Fig. 2. Original VHDL code

[15] Y. Sakabe, M. Soshi, and A. Miyaji, "Java obfuscation with a theoretical basis for building secure mobile agents," in *Proc. 7th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS2003), Lecture Note in Computer Science vol.2828*, 2003, pp. 89–103.

[16] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," in *Tech. rep. 148, Dept. of Computer Science, University of Auckland*, 1997.

[17] K. Fukushima, S. Kiyomoto, and T. Tanaka, "Obfuscation scheme using exclusive-or encoding," in *Proc. of 7th International Workshop on Information Security Applications (WISA2006)*, 2006, pp. 453–467.

[18] M. Brzozowski and V. N. Yarmolik, "Obfuscation as a intellectual protection in VHDL language," in *Proc. of 6th International Conference on Computer Information Systems and Industrial Management Applications (CISIM'07)*, 2007, pp. 337–340.

[19] R. Stern, P. Maciaszek, A. G. Michael Hsia, and R. Karri, "Digital logic obfuscation techniques to thwart cloning of asics," http://isis.poly.edu/csaw/winners/research/RichardStern.pdf.

[20] SEMANTIC DESIGNS, INC., "VHDL Source Code Obfuscator," http://www.semdesigns.com/Products/Obfuscators/VHDLObfuscator.html.

[21] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," *Journal of Cryptology*, vol. 3, no. 2, pp. 99–111, 1991.

[22] National Institute of Standards and Technology, "AES key wrap specification," http://csrc.nist.gov/groups/ST/toolkit/documents/kms/key-wrap.pdf, 2001.

[23] RSA Laboratories, "PKCS#1 v2.1: RSA Cryptography Standard," ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf, 2002.

[24] M. Bellare and P. Rogaway, "Optimal asymmetric encryption," in *Proc. of EUROCRYPT'94, Lecture Note Computer Science 950*, 1995, pp. 92–111.

[25] H. Berghel, "Watermarking cyberspace," *Communications of ACM*, vol. 40, no. 11, pp. 19–24, 1997.

[26] S. Craver, N. Memon, B. Yeo, and M. Yeung, "Resolving rightful ownerships with invisible watermarking techniques: Limitations, attacks, and implications," *IEEE Journal on Seleclted Areas in Communications*, vol. 16, no. 4, pp. 573–586, 1998.

[27] C. Collberg, "Csc 620 security through obscurity. course notes." http://www.cs.arizona.edu/~collberg/Teaching/620/2002/Handouts/Handout-13.pdf, 2002.

[28] zynamics.com, "zynamics BinDiff," http://www.zynamics.com/bindiff.html.

Appendix

We provide the watermarked VHDL code using our proposed mechanism based on a software obfuscation technique. The original code for an 8-bit counter is presented in Figure 2. A watermark is naively embedded in variable `dummy` in Figure 3. However, the value of this variable is not used. Thus, an illegal user may know that this variable is included only to embed a watermark. Figure 4 shows the obfuscated code where the variables `dummy` and `counter` are encoded in `D` and `C`. These variables are encoded

```
counter:process (clock);
  begin
      - dummy variable -
    dummy <= "10010010";
    count <= "0";
    if (clock'event and clock = "1") then
      count <= count + 1;
    end if;
  end process;
```

Fig. 3. Watermark embedded in dummy variable

```
counter:process (clock);
  begin
    D <= "10111101";
    C <= "10110100";
    if (clock'event and clock = "1") then
      C <= D xor ((D xor C xor "10011011") + 1)
          xor "10011011";
      D <= D xor C xor "00101111";
    end if;
  end process;
```

Fig. 4. Obfuscated VHDL code

according to the rule:

$$\begin{pmatrix} \texttt{D} \\ \texttt{C} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \texttt{dummy} \\ \texttt{counter} \end{pmatrix} \oplus \begin{pmatrix} 00101111 \\ 10110100 \end{pmatrix}.$$

The value of the variable `dummy` is mixed with that of the existing variable counter. A malicious user would have difficulty in ascertaining that the code contains a redundant variable since the values of both `D` and `C` are used to calculate the next value.