IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON MOBILE SERVICE COMPUTING WITH INTERNET OF THINGS

Received January 7, 2019, accepted January 22, 2019, date of publication January 30, 2019, date of current version April 5, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2896284

Obfuscation-Based Watermarking for Mobile Service Application Copyright Protection in the Cloud

SUN GUANG[1,2], F. XIAOPING[1,3], J. WANGDONG[1], L. FENGHUA[1], AND J. YUEWEI[1]

[1]Hunan University of Finance and Economics, Changsha 410205, China
[2]The University of Alabama, Tuscaloosa, CO 35401, USA
[3]Central South University, Changsha 410012, China

Corresponding author: Sun Guang (simon5115@163.com)

This work was supported in part by the Open Foundation for the University Innovation Platform in the Hunan Province under Grant 16K013, in part by the Hunan Provincial Natural Science Foundation of China under Grant 2017JJ2016, in part by the 2016 Science Research Project of Hunan Provincial Department of Education, under Grant 16C0269, and in part by the Accurate Crawler Design and Implementation with a Data Cleaning Function, National Students Innovation and Entrepreneurship of Training Program, under Grant 201811532010, and in part the 2011 Collaborative Innovation Center for Development and Utilization of Finance and Economics Big Data Property, Universities of Hunan Province.

ABSTRACT The contributions of cloud computing in the prevention of software piracy are inadequate, and there are still rampant piratical mobile service applications in the cloud. This paper navigates mobile service application copyright protection in the cloud and sets a watermarking example to explain it. We use Monden's method to obfuscate the application's source code, remove a part of the semantics, and add it to a recovery module. Because these obfuscation rules come from watermarks, the watermarks are mapped into the rules. The recovery module is a recognizer to prove the watermarks when the original program is recovered. The experimental results indicate that the obfuscated code becomes difficult to reverse engineering and the watermarks are robust.

INDEX TERMS Cloud computing, application copyright protection, watermarking, code obfuscation.

I. INTRODUCTION

Many people have assumed that the Software-as-a-Service (SaaS) model can completely solve the problem of software application piracy in the cloud [1], [2]. That is because the attacker and the codes are entirely separated by the cloud, which should undermine the foundation of software application piracy [3], [4]. However, according to a report issued by the BSA, from 2015 to 2017 the worldwide unlicensed software rate was 37 percent, and the commercial value of unlicensed software was $46.3 billion globally. What requires more attention is that the IDC estimates that the cloud now delivers 22 percent of software functionality worldwide [1]. The rate of properly licensed software and the corresponding losses are almost equal to the traditional computing environment [5],[6],[7]; that is, the contribution of cloud computing in the prevention of piracy is very small, and there still exists rampant piratical software applications in the cloud. The contribution of cloud computing in the prevention of software piracy is inadequate.

Software watermarking pertains to software application protection technology, which uses embedded information to provide application copyrights and deter software piracy [8]. For the past few years, software watermarking has achieved a large number of research results and has been proven to be an effective software protection technology [9], [10]. However, currently, this technology is out of date and focuses on the traditional computing environment [11]–[13]. The behaviors of software application piracy in the cloud are ‘‘new’’ to us [14], [15]. The software watermarking technology should be set in the ‘‘new’’ platform for development. In the cloud, software watermarking should face appropriate cloud-based situations such as an application installed on a server that is remote to users and uncontrolled [16]. Applications such as mobile service applications in the cloud are very sensitive to their codes, and addressing the algorithm's robustness and code semantic hiding is complicated [17], [18]. The watermarking approach presented in this paper uses obfuscated

The associate editor coordinating the review of this manuscript and approving it for publication was Tie Qiu.

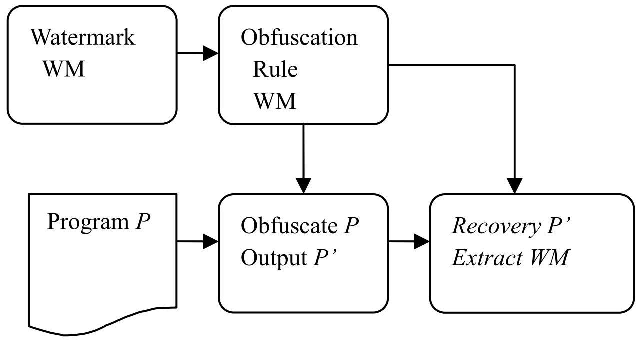


FIGURE 1. Framework of obfuscation-based software watermarking.

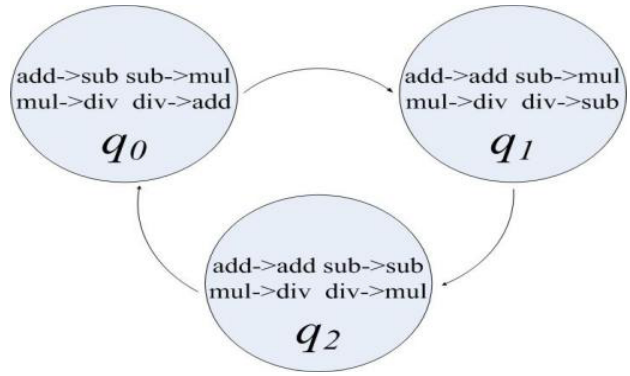


FIGURE 2. Example of watermark obfuscation and insertion.

interpretation to map the watermarks to obfuscation rules to hide the code semantics and prove copyright at the same time.

II. OBFUSCATION AND WATERMARK EMBEDDING

A framework of the software watermarking is shown in Figure 1. The obfuscation scheme we use in this paper was proposed by A. Monden, A. Monsifrot, and C. Thomborson [19]–[21]. Watermark information WM consists of numbers with radix R, and the obfuscation rule is created with the watermark information. According to the obfuscation rule, the program is obfuscated. The approach obfuscates the codes presented, and at the same time, the watermark information is mapped to the obfuscation rule. The obfuscated code is not equivalent in function to the original code; it cannot directly execute the obfuscated program before the semantics are recovered. Semantic recovery requires a single module that also performs watermark verification.

The definition of our obfuscation is as follows:

Given a program p and a code transform T which inputs p and outputs p', it can be said that T is a nonequivalent semantic obfuscation of p if it meets the following conditions.

1. If p' and p have the same input, the output is different;

2. The readability of p' is far below p; in other words, the cost of reverse engineering p' is much stronger than p;

3. There exist no converse transformations from p' to p, or it is difficult to construct a tool to finish this converse transformation, or the cost is very high.

In nature, T is a sequential process, there are no concurrent, competitive, synchronous operations. T is a typical FSM (finite state machine) that can be defined a by 6-tuple $(Q, \sum, \Psi, \Delta, \Lambda, q_0)$, where:

$Q = \{q0, q1, \cdot\cdot, qm-1\}$ is a state set of FSM, where m is equal to or greater than the length of WM, and q_0 is the initial state.

$\sum = \{C_0, C_1, \ldots, C_{n-1}\}$ is a set of inputs, each element is an instruction of p, and n is the number of instructions.

$\psi = \{Ct_0, Ct_1, \ldots, Ct_{n-1}\}$ is a set of outputs, $\sum = \Psi$, that is, p and p' belong to the same programming language.

$\delta i : \Sigma \to Q$, is a state transition function of q_i to define how to transform a certain state q_i.

$\Delta = \{\delta 0, \delta 1, \ldots\ldots, \delta l - 1\}$ is a set of state transition functions, $l = m * n$.

$$Q = \{q0, q1, q2\}, \tag{1}$$

$$\sum = \psi = \{add, sub, mul, div\}, \tag{2}$$

$$\delta 0(add) = \delta 0(sub) = \delta 0(mul) = \delta 0(div) = q1; \tag{3}$$

$$\delta 1(add) = \delta 1(sub) = \delta 1(mul) = \delta 1(div) = q2; \tag{4}$$

$$\delta 2(add) = \delta 2(sub) = \delta 2(mul) = \delta 2(div) = q0; \tag{5}$$

$$\Delta = \{\delta 0, \delta 1, \delta 2\}, \tag{6}$$

The transition function λ_i is defined as WM,

$$WM = w_0 w_1 \ldots\ldots w_k, \quad 0 \le w_i < R, \tag{7}$$

$\lambda i : \sum \to \psi$, is a code transition function of q_i to define how to obfuscate an instruction $c_i \Lambda = \{\lambda 0, \lambda 1, \ldots, \lambda l - 1\}$.

$\Lambda = \{\lambda 0, \lambda 1, \cdot\cdot, \lambda l - 1\}$, is a set of code transition functions.

Figure 2 shows a small example of nonequivalent semantic obfuscation that has three statuses and four instructions.

As Figure 2 shows, if we insert a watermark "012" into the codes, we can define:

$$\lambda 0(add) = sub; \quad \lambda 0(sub) = mul; \quad \lambda 0(mul) = div; \quad \lambda 0(div) = add; \quad \lambda 1(add) = add; \quad \lambda 1(sub) = mulx; \tag{8}$$

$$\lambda 1(mul) = div; \quad \lambda 1(div) = sub; \quad \lambda 2(add) = add; \quad \lambda 2(sub) = sub; \quad \lambda 2(mul) = div; \quad \lambda 2(div) = mul; \tag{9}$$

All instructions were transformed differently with function λ_0, one instruction was kept in λ_1, and two instructions were kept in λ_2. $\Lambda = \{\lambda 0, \lambda 1, \lambda 2\}$.

According to our obfuscation framework, inputs add, add, sub, div, mul, add, mul, will output the series of sub, add, sub, add, div, add, div.

The obfuscation obeys the following constraints.

1. $\delta i : \Sigma \to Q$ is a bijection; its inverse is $\delta_i^{-1} : Q \to \Sigma$.

2. $\lambda i : \sum \to \psi$ has the corresponding bijection relation, $\lambda i^{-1} : \psi \to \sum$.

3. Divide $\delta_i^{-1} : Q \to \Sigma$ into several subsets, put the same operation number commands into an identical subset, prescribe a constraint in which the independent variable and dependent variable come from identical subsets in function $\delta_i^{-1} : Q \to \Sigma$.

III. SEMANTIC RECOVERY AND WATERMARKS EXTRACTION

Recovering the semantic cannot set up a decompiler δ_i^{-1} : $Q \rightarrow \Sigma$ and λi^{-1} : $\psi \rightarrow \sum$. For example, the following codes implement computation p:= 1+2+3+...+n. Clearly, this requires a loop statement.

```
let x = n
let p = 0
Loop: if x == 0
then exit
else
add p,x
sub x,1
goto loop
endif
```

The FSM performs the obfuscation as follows:

$$\lambda_0(add) = mul, \quad \lambda_1(sub) = div,$$
$$\delta_0(add) = q1, \quad \delta 1,(sub) = q2, \ \delta 1(mul) = q0, \quad (10)$$

if() then else endif, goto, do not set this transform instruction, q0 is the initial state, and the obfuscated codes are as follows:

```
let x = n
let p = 0
Loop: if x == 0
then exit
else
mul p,x
div x,1
goto loop
endif
```

The decompilation corresponding to the FSM is:

$$\lambda 0^{-1}(mul) = add; \quad \lambda 1^{-1}(div) = sub;$$
$$\delta 0^{-1}(mul) = q1; \quad \delta 1^{-1}(div) = q2; \quad (11)$$
$$\lambda 2^{-1}(mul) = div; \quad \lambda 3^{-1}(div) = sub;$$
$$\delta 2^{-1}(mul) = q3; \quad \delta 3^{-1}(div) = q4 \quad (12)$$

Regarding the output of FSM as input, the decompiler will output

```
let x = n
let p = 0
Loop: if x == 0
then exit
else
mul p,x
div x,1
goto loop
endif
```

According to the output codes, the initial state is q_0, and while the final state is q_2 in the first loop, in the second loop, the initial state is q_2. Obviously, the result of translating the same loop body code into different semantics must be incorrect because the initial state is different. It is essential to obtain the correct result and that the initial state is identical each time the decompiler executes the same loop body. There are two methods to achieve this result:

$$1. \ \lambda_0 = \lambda_1 = \lambda_1 = \ldots\ldots = \lambda_{l-1}$$
$$2. \ \delta(\textit{first instruction of loop}) \quad (13)$$
$$= \delta(\textit{final instruction of loop}) \quad (14)$$

Method 1 cancels the state transition, and method 2 converts the state of the loop body into a return circuit to ensure that the initial states are identical. Our study uses Akito Monden's "dummy instruction" method to achieve the conversion of the state. In the following codes, the last instruction mul x, 1 is nonsense; it only converts the last state to q_0.

Let us suppose that the FSM achieved the nonequivalent semantic obfuscation is W_x, the decompiler is W_x^{-1}, and it is necessary to set up W_x^{-1} first to recover the obfuscated codes and then insert the dummy instructions in the loop body.

W_x^{-1} is also a 6-tuple (Q', $\sum'$, Ψ', Δ', Λ', q_0), where:

$Q' = \{q0, q1, \ldots\ldots, qm' - 1\}$ is the set of states in W_x^{-1};

$\sum' = \Psi$ is the set of inputs to W_x^{-1};

$\Psi' = \sum$ is the set of outputs from W_x^{-1};

$\Delta' = \{\delta'0, \delta'1, \ldots\ldots, \delta'l' - 1\}$ is the set of state transition functions in W_x^{-1};

$\Lambda' = \{\lambda'0, \lambda'1, \ldots\ldots, \lambda'l' - 1\}$ is the set of translation functions in W_x^{-1}.

$q0$ is the initial state; $m' = m$, $l' = l$.

$\forall$i, $j\delta' i(cj) = \delta i(\lambda i(cj))$, namely, when W_x^{-1} inputs C_j under the q_i, the output is equal to the W_x input $\lambda i(cj)$ under the q_i.

Figure 3 shows the W_x^{-1} built up by the W_x as follows:

$$Q' = Q = \{q_0, q_1, q_2\}, \quad (15)$$
$$\sum{}' = \Psi = \{add, sub, mul, div\}, \quad (16)$$
$$\delta'0(add) = \delta'0(sub) = \delta'0(mul) = \delta'0(div) = q_1; \quad (17)$$
$$\delta'1(add) = \delta'1(sub) = \delta'1(mul) = \delta'1(div) = q_2; \quad (18)$$
$$\delta'2(add) = \delta'2(sub) = \delta'2(mul) = \delta'2(div) = q_0; \quad (19)$$
$$\Delta' = \{\delta_0', \delta_1', \delta_2'\}, \quad (20)$$
$$\psi' = \sum = \{add, sub, mul, div\}; \quad (21)$$

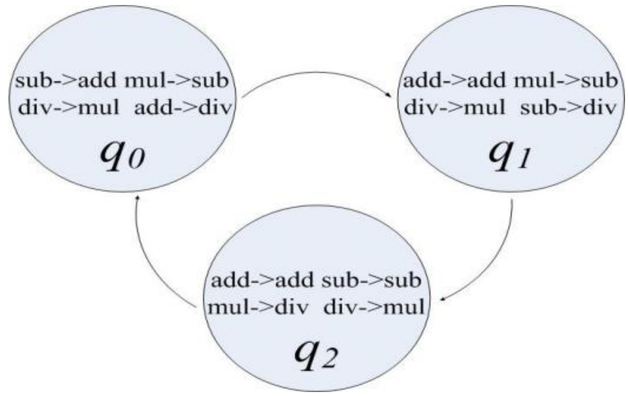


FIGURE 3. Example of W_x^{-1}.

$$\lambda'_0(sub) = add; \quad \lambda'_0(mul) = sub;$$
$$\lambda'_0(div) = mul; \quad \lambda'_0(add) = div; \tag{22}$$
$$\lambda'_1(add) = add; \quad \lambda'_1(mul) = sub;$$
$$\lambda'_1(div) = mul; \quad \lambda'_1(sub) = div; \tag{23}$$
$$\lambda'_2(add) = add; \quad \lambda'_2(sub) = sub;$$
$$\lambda'_2(mul) = div; \quad \lambda'_2(div) = mul; \tag{24}$$

q_0 is also the initial state of W_x^{-1}. The model that corresponds with W_x^{-1} is shown in Figure 3. W_x^{-1} cannot restore all of the semantics; because of the loop instruction, the loop body requires the identical initial state to translate. That is, for any state among $< qi, qj >$, there exists an instruction sequence $< d1, d2 \ldots\ldots, dh >$ in the $\exists$; if h is greater than or equal to 1, q_i converts to q_j, while W_x^{-1} is input to the sequence below under the q_i state. For this purpose, some constraints are added to W_x^{-1}:

The state conversion chart of W_x^{-1} is connected, and the shortest path of the overall situation is the shortest.

All branches of the same instruction must have the same final state.

The watermark extraction of this scheme was realized through testing and verified. An instruction sequence is constructed to traverse all states of W_x or W_x^{-1}, accessing each state in turn, and the unchanged instruction number of the set under the $q_i$is equal to the value of ith bit position in WM.

IV. EXPERIMENT AND ANALYSIS

This experiment aims to test the decompilation from p_x to p verify the intensity of the obfuscation and the robustness of the watermark.

We used the Java program TicTacToe in the experiment, and 200 successive instructions were selected randomly as a segment. The obfuscation rules are listed in Table 1, with a maximum of 8 states and 8 instructions transformed in each state. For any state, the number n of the transformed instructions means that, in this state, only n instructions are transformed. There are 213 instructions in the Java Jasmine format, so the inserted watermark is

$$WM = (213-8)(213-5)(213-8)(213-5)(213-8)$$
$$(213-5)(213-6)(213-5) \tag{25}$$

We selected 20 decompiler tools (see Table 2) to test the decompilation of the obfuscated program. The decompilation results are shown in Table 2. "before" represents the original program p, "after" represents program p', the symbol "×" indicates that the.java document cannot be generated while decompiling. "△" indicates that the.java document can be generated but the recompilation failed, "□" indicates that both the decompilation and recompilation were successful, but the program after recompilation cannot be executed correctly because the executive program and the original program are nonequivalent in the function or the program resources are not matched. "◯" shows that the decompilation and the watermarks are exacted, in addition, the input of the restored program is equal to the output of the original program. Maybe the decompiled program is not good enough at the code level, but the input and the output are the same; we judge that the decompilation was successful. Once the decompilation works, the code hiding of p fails,

TABLE 1. Input/Output of the test program.

STATE	INPUT/OUTPUT	STATE TRANSFORM
q0	goto / if_icmpne	q4
	if_icmpne / goto	q7
	iload_1 / iconst_2	q5
	iconst_1 / iload_1	q2
	iconst_2 / iconst_1	q1
	iadd / irem	q6
	iload_2 / iadd	q3
	irem / iload_2	q0
q1	iload_1 / iconst_1	q1
	iconst_1 / iconst_2	q3
	iconst_2 / iload_1	q4
	iadd / iload_2	q2
	iload_2 / irem	q0
q2	goto / if_icmpne	q7
	if_icmpne / goto	q0
	iload_1 / iload_2	q6
	iconst_1 / iconst_1	q5
	iconst_2 / iconst_2	q4
	iadd / imul	q3
	iload_2 / iload_1	q1
	irem / idiv	q2
q3	iload_1 / iconst_1	q3
	iconst_1 / iadd	q6
	iconst_2 / irem	q2
	iadd / iload_2	q5
	iload_2 / iconst_2	q1
q4	goto / if_icmpne	q5
	if_icmpne / goto	q4
	iload_1 / iconst_2	q2
	iconst_1 / irem	q1
	iconst_2 / iload_2	q3
	iadd / iload_1	q0
	iload_2 / iconst_1	q7
	irem / iadd	q6
q5	iload_1 / irem	q0
	iconst_1 / iload_2	q1
	iconst_2 / iload_1	q4
	iadd / iconst_2	q2
	iload_2 / iadd	q3
q6	goto / if_icmpne	q2
	if_icmpne / goto	q5
	iload_1 / iload_1	q4
	iconst_1 / iconst_1	q1
	iconst_2 / iconst_2	q6
	iadd / iadd	q7
q7	iload_1 / iconst_1	q3
	iconst_1 / iadd	q4
	iconst_2 / iload_2	q1
	iadd / irem	q7
	iload_2 / iload_1	q0

TABLE 2. Results of the decompile test.

Decompiler	Before	After	Watermark
Number of obfuscated instructions	200	213	
Ocha	*O*	□	exist
SourceTec Decompiler	*O*	□	exist
Jad	*O*	△	exist
Front End Plus	*O*	△	exist
DeJava	*O*	△	exist
Decafe Pro	*O*	△	exist
CavajJava Decompiler	*O*	△	exist
DJ Java Decompiler	*O*	△	exist
NMI's Java Class Viewer	*O*	△	exist
JReversePro	*O*	□	exist
JODE	*O*	×	exist
JCavajJava Decompiler	*O*	×	exist
HomeBrew Decompiler	*O*	×	exist
Dava Decompiler	*O*	○	broken
Jshrink	*O*	△	exist
Class Spy	*O*	×	exist
jAscii	*O*	△	exist
ClassCracker	*O*	×	exist
SourceAgain	*O*	×	exist
WingDis	*O*	×	exist

even if we can prove the watermarks, the schema is considered a fail.

We can see in Table 2 that originally, there are 200 instructions, while there are 213 instructions in the obfuscated program. The extra 13 instructions are dummy instructions. All the decompiler tools decompiled p. However, for p', JODE, the JCavajJava Decompiler, the HomeBrew Decompiler, Class Spy, ClassCracker, SourceAgain and WingDis failed to generate the standard.Java document. In addition, Jad, Front End Plus, DeJava, Decafe Pro, CavajJava Decompiler, DJ Java, Decompiler, NMI's Java Class Viewer, Jshrink, and jAscii failed to recompile. Although the decompilation and recompilation of ocha, the SourceTec Decompiler, and JreversePro were successful, the program could not run after the recompilation. Only the Dava Decompiler successfully decompiled the testing code, and the recompiled program was equal to the original program in function. This tool uses the Exhaustive Attack method to decompile and no surprise to crack the obfuscation.

To compare the applicability in the cloud between this study and the existing watermark algorithms, the experiment tested the existence of watermarks with existing algorithms after obfuscating the nonequivalent semantic. Cloud users usually require the code semantics to be hidden; if the algorithm *ali* can extract the watermark information successfully after the obfuscation, this means that the *ali* has the same ability to prove the copyright as this scheme after hiding the semantics. The watermark robustness of *ali* was not lower than this scheme in the cloud environment. The experiment tested 10 algorithms that are available at Sandmark, and the program under test is in accord with the previous experiment. The embedded information has the same watermarks. The experimental results are shown in Table 3.

TABLE 3. Adoption test of software watermarking algorithms.

Algorithm Abbreviation	Watermark Technology	Embedding Watermark	Extracting Watermark
W1	StringConstant	failure	failure
W2	Stern	success	failure
W3	RegisterType	success	failure
W4	Qu/Potkonjak	success	failure
W5	Monden	success	failure
W6	Graph	success	failure
W7	AddMethField	success	failure
W8	AddSwitch	success	failure
W9	Addinitialization	failure	failure
W10	AddExpression	success	failure

The results show that, with the exception of W1 and W9, each algorithm successfully embedded the watermark information. W1 and W9 are static software watermarking algorithms. W1 embedded the watermark by adding or changing the constant definition, and W9 achieved watermarking by utilizing the initialization. The watermark information of the two algorithms are both embedded in the data segment, and there is no need to modify the code. The sample case of this experiment only contains the executable code, so these two watermarking algorithms cannot be implemented. According to Collberg's experiment, the two algorithms cannot handle any obfuscated transition, and they have the worst robustness. There is no algorithm that can extract the watermark after the obfuscation, which demonstrates that these algorithms cannot reach the robustness in this scheme.

V. CONCLUSIONS

Some people believe that the SaaS model greatly minimizes software piracy because, after migrating programs from the desktop to the Cloud, the Cloud separates the adversary with source codes; software piracy will be more difficult to accomplish. SaaS applications continuously require an Internet connection and are able to gain basic control over their applications, and thus, most likely solving the piracy problem on its own.

We argue that the outsider threat to software copyright can be effectively solved by the Cloud's separation. However, insider threats are more difficult to address. Who guards the guards? We currently have no way to monitor the security arrangements in the Cloud. Software distributions in the Cloud are supposed to mark the beginning of new concerns about security breaches, piracy abuses and access control violations.

This study understands the new challenges of mobile service application copyright protection in the Cloud. The goal of our software watermarking does not change with the Cloud; the "how" of software watermarking must be considered for the Cloud environment. We use the nonequivalence of semantic obfuscation to hide code semantics and embed watermark information at the same time. We intend to watermark the program and obfuscate the codes as well. According to the decompilation test, only the exhaustive attack method can crack the 200 instructions under 8 state transition schemes, which indicates that the scheme can address watermark destruction.

REFERENCES

[1] Cloud Security Alliance (CSA). *Top Threats to Cloud Computing: Deep Dive*. Accessed: Aug. 2018. [Online]. Available: https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-deep-dive/

[2] Z. Yu, C. Wang, C. Thomborson, J. Wang, S. Lian, and A. V. Vasilakos, "A novel watermarking method for software protection in the cloud," *Softw.-Pract. Exper.*, vol. 42, no. 4, pp. 409–430, 2012.

[3] H. Keiko, G. R. David, F.-M. Eduardo, and B. F. Eduardo, "An analysis of security issues for cloud computing," *J. Internet Services Appl.*, vol. 4, no. 10, pp. 109–114, 2013.

[4] D. G. Rosado, R. Gómez, D. Mellado, and E. Fernández-Medina, "Security analysis in the migration to cloud environments," *Future Internet*, vol. 4, no. 2, pp. 469–487, 2012.

[5] J. Chen, K. Li, W. Wen, W. Chen, and C. Yan, "Software watermarking for java program based on method name encoding," in *Advanced Intelligent Systems and Informatics*. 2017, pp. 865–874.

[6] Z. Chen, C. Jia, and D. Xu, "Hidden path: Dynamic software watermarking based on control flow obfuscation," in *Proc. Int. Conf. Comput. Sci. Eng. (CSE) Embedded Ubiquitous Comput. (EUC)*, vol. 2, Jul. 2017, pp. 443–450.

[7] Z. Chen, Z. Wang, and C. Jia, "Semantic-integrated software watermarking with tamper-proofing," *Multimed Tools Appl.*, vol. 77, no. 9, pp. 11159–11178, 2017.

[8] S. Guang, X. Fan, S. Fu, Y. Song, and L. Huifang, "Software watermarking in the cloud: Analysis and rigorous theoretic treatment," *J. Softw. Eng.*, vol. 9, no. 2, pp. 410–418, 2015.

[9] Y. Awasthi, P. R. Agarwal, and K. B. Sharma, "Intellectual property right protection of browser based software through watermarking technique," *Int. J. Comput. Appl.*, vol. 97, no. 12, pp. 32–36, 2014.

[10] M. D. Preda and M. Pasqua, "Software watermarking: A semantics-based approach," *Electron. Notes Theor. Comput. Sci.*, vol. 331, pp. 71–85, Mar. 2017.

[11] D. A. B. Fernandes, L. F. B. Soares, J. V. Gomes, M. M. Freire, and P. R. M. Inácio, "Security issues in cloud environments: A survey," *Int. J. Inf. Secur.*, vol. 13, no. 2, pp. 113–170, 2014.

[12] G. Hurel, R. Badonnel, A. Lahmadi, and O. Festor, "Outsourcing mobile security in the cloud," in *Monitoring and Securing Virtualized Networks and Services*. Berlin, Germany: Springer, 2014, pp. 69–73.

[13] Y. Wang, D. Gong, B. Lu, F. Xiang, and F. Liu, "Exception handling-based dynamic software watermarking," *IEEE Access*, vol. 6, pp. 8882–8889, 2018.

[14] N. Zong and C. Jia, "Software watermarking using support vector machines," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, vol. 2, Jul. 2015, pp. 533–542.

[15] X. Rao and V. K. N. Lau, "Distributed fronthaul compression and joint signal recovery in cloud-RAN," *IEEE Trans. Signal Process.*, vol. 63, no. 4, pp. 1056–1065, Feb. 2015.

[16] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[17] M. Chen, Y. Hao, L. Hu, K. Huang, and V. Lau, "Green and mobility-aware caching in 5G networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 12, pp. 8347–8361, 2017.

[18] L. Zhou, "QoE-driven delay announcement for cloud mobile media," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 1, pp. 84–94, Jan. 2017.

[19] A. Monden, A. Monsifrot, and C. Thomborson, "A framework for obfuscated interpretation," in *Proc. 2nd Workshop Australas. Inf. Secur. Data Mining Web Intell., Softw. Internationalisation*, vol. 32, 2004, pp. 7–16.

[20] S. Patel and T. Pattewar, "Software birthmark based theft detection of JavaScript programs using agglomerative clustering and improved frequent subgraph mining," in *Proc. Int. Conf. Adv. Electron. Comput. Commun.*, Bangalore, India, Oct. 2014, pp. 1–6.

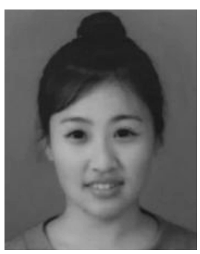
[21] Z. Tian, Q. Zheng, T. Liu, M. Fan, E. Zhuang, and Z. Yang, "Software plagiarism detection with birthmarks based on dynamic key instruction sequences," *IEEE Trans. Softw. Eng.*, vol. 41, no. 12, pp. 1217–1235, Dec. 2015.

SUN GUANG received the Ph.D. degree in computer science from Hunan University, Changsha, Hunan, China, in 2012. He is currently a Professor with the Institute of Big Data, Hunan University of Finance and Economics, Changsha. He is also a Visiting Scholar with The University of Alabama. His research was supported by the Open Foundation for the University Innovation Platform from the Hunan Province, China, under Grant 16K013. His research interests include the umbrella of sensor networks security, information hiding (with a focus on software watermarking and software birthmarking), and big data analysis and visualization.

F. XIAOPING received the Ph.D. degree. He was also a Professor, a Doctoral Tutor, and an Expert with special government allowances from the State Council, top talents in the Ministry of Railways, and cross-century academic and technical leaders in Hunan, selected for the first batch of talent projects in the new century in Hunan. He was the Director of the Academic Affairs Office, Changsha Railway Institute, the Associate Dean of the School of Information Engineering, and the Director of the Automation Engineering Research Center. He is the Director of the Office of the Degree Evaluation Committee, Central South University (Director of the Graduate School of Academic Degrees), where he is also the Director of the Network System Research Institute, and a Professor and a Doctoral Tutor with the School of Information Science and Engineering. He is currently the Vice President of Hunan Finance and Economics College.

J. WANGDONG was born in Yangzhou, Hunan, China, in 1971. He received the B.S. degree in mathematics from Hunan Normal University, in 1993, and the M.S. degree in computer applications from Guangxi Normal University, in 2005. From 2005 to 2014, he was an Associate Professor with the Information Management Department. Since 2015, he has been an Associate Professor with the Institute of Big Data, Hunan University of Finance and Economics. His research interests include machine learning and data mining for big data.

L. FENGHUA was born in Hengyang, Hunan, China, in 1997. From 2017 to 2018, she was a Student Assistant with the school's innovation training laboratory. She has presided over the national college student innovation and entrepreneurship training program project accurate crawler design and implementation with data cleaning function, and she has participated in the college project approval. Since 2015, she has been studying information management and information systems with the Hunan University of Finance and Economics. Her research interests include data visualization applications, computer programming, data analysis, data collection, and other big data applications. She has received the Second Prize from the Hunan University Student Computer Programming Competition, in 2017, and the Bronze Prize from the Hunan University Students Entrepreneurship Competition, in 2018.



J. YUEWEI was born in Changchun, Jilin, China, in 1997. She is currently pursuing the B.S. degree in information management and systems from the Hunan University of Finance and Economics. At school, she helped to finish this paper. Meanwhile, she has published an article in the *Journal of Lanzhou Institute of Technology*. In addition, many of her works have been published in the school newspaper.