Vol 8. No. 4, December, 2015
ISSN 2006-1781

African Journal of Computing & ICT

www.ajocict.net

A Comparative Analysis of Static Java Bytecode Software Watermarking Algorithms

K. Kumar[1], V. Kehar[2] & P. Kaur[3]
[1]Faculty of Science and Technology, ICFAI Baddi, INDIA
[2] Department of Computer Science & Engineering, National Institute of technology, Hamirpur ,INDIA
[3]Dept. of Computer Engg. & Technology, Guru Nanak Dev University, Amritsar,INDIA

ABSTRACT

Software Piracy is one of the biggest problem faced by software industry causing millions of dollars loss every years to the software developing companies. The global revenue loss was estimated to be more than $62.7 billion in year 2013 due to the software piracy. Software watermarking techniques which attempts to protect the software by embedding copyright notice or unique identifiers into software to prove the ownership of software. Software Watermarking discourage piracy; as a proof of purchase or authorship; also helps in tracking the source of illegal redistribution of copies of software. We have compared and analyzed the existing watermarking algorithms by using them to watermark Java jar files and then applying the distortive attacks to each watermarked program by applying obfuscation and optimizing. After studying the results obtained, we found that high proportion of embedded watermarked were removed as results of transformation applied.

Keywords: obfuscation; watermarking; software piracy; Java bytecode; Software Protection;

African Journal of Computing & ICT Reference Format:

K. Kumar, V. Kehar & P. Kaur (2015): A Comparative Analysis of Static Java Bytecode. Software Watermarking Algorithms. Afr J. of Comp & ICTs. Vol 8, No. 3. Pp 201-208.

1. INTRODUCTION

From the last decade, code of the software is distributed in an architecturally-neutral format which has increased the ability to reverse engineer source code from the executable. With the availability of large amount of reversing tools on internet, it had become easy for crackers or/ and reverse engineer to copy, decompile and disassembling of software especially which are made from Java and Microsoft's common intermediate language as they are mostly distributed through internet. Many of the Software protection techniques can be reversed using the model described in [5].

As per Business Software Alliance (BSA) report [1], the commercial value of pirated software is $62.7 billion in year 2013. The rate of pirated software had been increased from 42 percent in 2011 to 43 percent in 2013 and in most of the emerging economies this rate is high. So, Software protection has become an important issue in current computer industry and become a hot topic for research [3, 4]. One of the technique to prevent the software piracy is software watermarking. *Software watermarking* is technique [2] used for embedding a unique identifier into an executable of a program. A watermark is similar to copyright notice; it asserts that you can claim certain rights to the program. The presence of watermark in program would not prevent any attacker from reverse engineering it or pirating it. However the presence of watermark in every pirated copy later will help you to claims the program is ours.

The embedded watermark is hidden in such a way that it can be recognized at later by using the *recognizer* to prove the ownership on pirated software [6]. The embedded watermark should be *robust* that it should be resilient to semantics preserving transformations. But it some cases it is necessary that watermark should be *fragile* such that it become invalid if the semantics preserving transformation are applied. This type of watermark is mostly suitable for the software licensing schemes, where if any change is made to the software which could disable the program.

Obfuscation and encryption are used for the purpose either preventing the decompilation or decreasing the program understanding, while fingerprinting and watermarking techniques are used to uniquely identify software to prove ownership. In this paper we present a survey of existing Java bytecode watermarking algorithms and performed a comparative analysis of static Java bytecode watermarking algorithms implemented in Sandmark [10] framework. Out of 14 Static watermarking algorithms we are going to compare the results obtained from 12 static watermarking algorithms. First section represents the details regarding the watermarking system, types, techniques etc., In second section evaluation of testing procedure, In third section we will presents the results of our research work and finally fourth section contains the results and future work.

Vol 8. No. 4, December, 2015
ISSN 2006-1781

2. BACKGROUND

Watermarking techniques are used extensively in the multimedia industry to identify the multimedia files such as video and audio files, and this concept has extended to software industry. The purpose of watermarking is not to make program harder as in case of obfuscation but it discourages the software thieves from illegal distributing copies of software as they know they could be identified [] .

2.1 Difficulties faced by Software watermarking

There are several problems are related with implementation of software watermarks and many of the current watermarking algorithms are vulnerable to attacks. Watermarking software system should meet the following conditions.

1. Program size: embedded watermarks should not increase the size of program significantly.
2. Program efficiency: efficiency of watermarked program or software must be similar to original program and need not be decreased significantly.
3. Robust watermarks: Embedded watermarks must be strong enough to distortive or semantics preserving transformations.
4. Embedded Watermarks must be well hidden, to avoid removal of watermark by the attacker.
5. Watermarks extraction process must by unique such that only software owner can extract the watermark.

One of the difficult problem which is need to solve is keeping the watermark hidden from adversaries while at the same time, allowing the software owner to efficiently extract the embedded watermark when needed. If it easy enough then an adversary would be able to extract watermark too. If the watermark is hidden well then software owner may have problem in extracting the watermark. Embedded software watermarks need to be efficient in several ways such as:Cost of embedding time, Cost of runtime, Cost of recognition time.

2.2 Watermarking techniques

Software watermarks can be divided into two categories: static and dynamic [11]. Static watermarks techniques embeds the watermark in the data/or code of the program while dynamic techniques embeds the watermark in a data structure built at runtime.

Static watermarks are embedded in the data and/or code of a program. For example, embed a copyright notice into the strings. In case of Java programs, watermarks could be embedded within their constant pool or method bodies of java class files. As before the academic research in the area of software watermarking started, some of pioneer static software watermarking techniques was presented in patents [11, 12]. The main problem with embedding a watermark as a string in program is that useless variables could also be easily removed by performing dead-code analysis, and most of the times when obfuscation or optimization of code is applied many useless method or variable names are either lost or renamed.

2.3 Types of watermark

Nagra et al. define four types of watermark [2,14]:

Authorship Marks are used to identifying a software author, or authors. It embeds an identification- mark related to owner in the cover object. These watermarks are mostly visible and robust to the attacks.

Fingerprinting Marks are used to serialize the cover object by embedding a different mark in every distributed copy. It is used to find the method or channel of distribution, i.e. the person who has illegally distributed the copies of software. The watermarks are mostly robust, invisible and consist of a unique identifier e.g. customer reference number.

Validation Marks are used by mostly end users to verify that software product is authentic, genuine and unchanged, for example in case of Java, digitally signed Java Applets. A common method is to compute the digest of software product and embeds into software as a watermark. A digest is computed by using the MD5 or SHA-1. A validation mark should be fragile and visible.

Licensing Marks are used to ensure the software is authenticate against a license key. One property of these marks are that they are fragile .The key should become useless if the watermark is damaged.

2.4 Types of Attacks to watermarks [2]

I. **Distortive attack:**this type of attack involves applying the semantics preserving transformations to a software, such as optimizations or obfuscations, thus removing any watermark which rely in program syntax.

II. **Additive attack:** In this attack, a new watermark is added by an attacker to the already watermarked program in order to cast doubt on which watermark was added first [7].

III. **Subtractive attack:** In this attack, an attacker decompiled or disassembled the code in order to remove the watermark from the program.

3. THE EMPIRICAL EVALUATION

We are going to evaluate and analyze static watermarking software techniques by watermarking the 35 [20] jarfiles with the existing watermarking algorithms implemented in Sandmark and then applying distortive attack to each watermarked Jar file by using obfuscation techniques. After all the Jar files have been transformed, we try to extract the embedded watermarks from the obfuscated jar files.It is possible that many watermarks will be lost during the obfuscations and attempt to find which obfuscations most affect the watermarks. We attempt to embed and recognize the watermark **GNDU-Asr** from the jar files.

3.1 The Watermarker

We are going to evaluate and analyze the 12 out of 14 watermarking algorithms implemented in SandMark [10]. SandMark is research framework developed Christian Collberg et al. at the University of Arizona for research in the area of software watermarking, code obfuscation, tamper-proofing of Java Bytecode.

3.1.1 Static watermarking algorithms are as:

1. **Add Expression:** this algorithm is very simple add a bogus expression containing the watermark to a class file.
2. **Add Initialization:** adds the bogus local variable to the different methods as a string into the Constant pool of a class file.
3. **Add Method and Field:**embeds the watermark by dividing a watermark into two parts, first partis stored in the name of a field, the second half store in the name of a method.
4. **Add Switch:** In this algorithm a watermark is embedded in the case values of a switch statement.
5. **Davidson/Myhrvold** [17]: watermark is embedded by re-ordering basic blocks present in program in a suitable method.
6. **Graph Theoretic Watermark** [23]: watermark is embedded in a control-flow graph of program, which is added to the original program.
7. **Monden** [41]: watermark is embedded by replacing opcodes in a dummy method, which is generated by Sandmark.
8. **Qu/Potkonjak** [38]: watermark is inserted in local variable assignments by adding constraints to the interference graphs.
9. **Register Types:** watermark is inserted by introducing local variables of certain Java standard library types.
10. **Static Arboit** [58-59]: is watermarking algorithm that embeds the watermark via opaque predicates. A watermark is encoded in an opaque predicate and then appending the predicate to a selected branch.
11. **Stern** [39]: watermark is embedded as a statistical object by creating a frequency vector representation of the code.
12. **String Constant:** inserts the watermark in a string of a random class.

3.2 The Transformation attacks

We are using distortive attacks to evaluate the watermarking algorithms. Sandmark research framework contains variety of Semantics Preserving obfuscation techniques which will be apply transformation to watermarking algorithms. We also use Proguard [25] to apply optimizations to test case programs. In total there are 37 different transformations to be applied.

3.3 The Test Case Jar files

All the test jar files are plugins for open source text editor jEdit [20]. These test Jar files are obtained by installing jEdit and then using built-in plugin manager to download the plugin Jar files.

4. RESULTS

4.1 Watermarking

As a results of embedding watermark, we have obtained 336 Jar files out of an expected 420 watermarked jars. There are some algorithms which failed to insert the specified watermark, which may be due to some error or incompatible program jar. For example String Constant, Add expression and Allatori managed to correctly embed watermarks in all 35 test Jar Program.It is about 80 % of the expected watermarked jar files were actually produced.

Vol 8. No. 4, December, 2015
ISSN 2006-1781
African Journal of Computing & ICT

www.ajocict.net

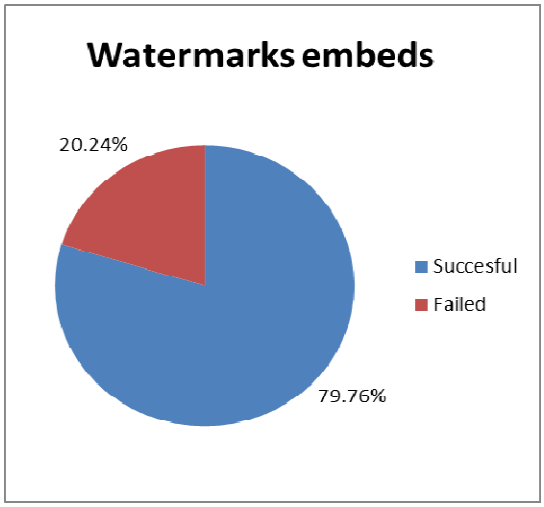

Table 1: shows the percentage of watermarks are embedded and failed.

	Total	Successful	Failed
Watermarks embeds	420	336	84
%age		79.76	20.24

Figure 1: depicts that around ~80% watermarks embeds and 20% gets failed due to some error or incompatible jar file. Out of the 336 watermarked jar files only 294 contained watermarks which were successfully recognized before the transformations attacks were applied. This means success rate is 87.5 % of the expected watermarked jar files produced actually recognized.

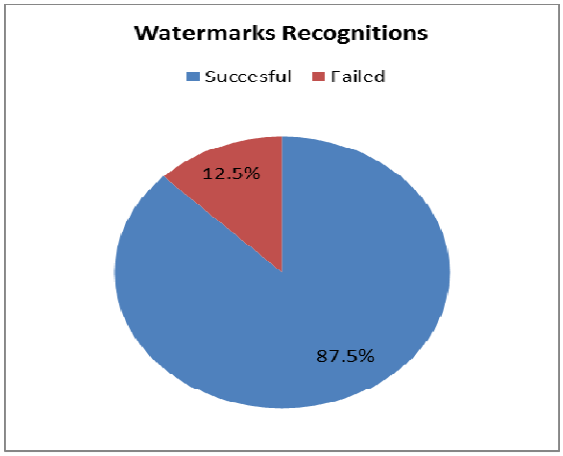

Table 2: Percentage of watermark recognition before the transformative attack applied.

	Total	Successful	Failed
Watermarks Recognitions	336	294	42
%age		87.5	12.5

Figure 2: depicts that 87.5% watermarks are recognized while 12.5% got failed before the transformative attacks are applied.

4.2 Obfuscation

We obfuscated 336 jar programs with36 obfuscation algorithms, and 1 optimization which should have resulted in 12432 attacked watermarked jars. There are some algorithms failed to output some jars so we actually obtained 11223 attacked watermarked jars using 37 semantics preserving transformations. This means that only 90.28 % attacked watermarked Jar files were actually produced.

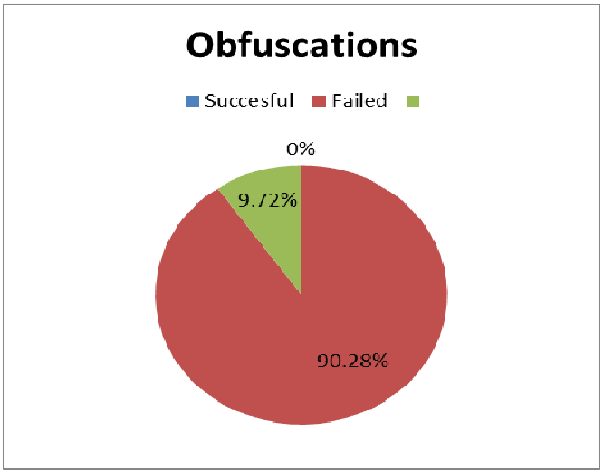

Table 3: percentage of jar file obtained after obfuscation is applied.

	Total	Successful	Failed
Obfuscation	12432	11223	1209
%age		90.28	9.72

Figure 3: depicts that we have obtained around 90% jar files after successfully applying the obfuscation to the watermarked jar files.

4.3 Recognition

Result of recognizing the watermark, embedded by different watermarking algorithms after applying the transformative attack i.e. after applying obfuscation, the resulting obtained watermarked jar files are shown by line graph. The horizontal bar line is marked with numbers indicating the number of successful recognitions of watermarks with respect to particular obfuscation algorithm.
A number of zeros can be seen throughout the graph indicating that no watermarks was recognized with that combination of transformation and watermark.

4.4 Analysis of Results

We have tested the static watermarking algorithms implemented within Sandmark with respect to *distortive attacks*. Distortive attacks are any semantics preserving code transformations, such as code obfuscation or optimization algorithms.
By examining the above figure it is found that many watermarks got lost due to obfuscations techniques applied.
Important observations of **Comparative analysis** are as:

i. Number of watermarks gets lost due to transformation applied by obfuscation algorithms.
ii. String constant watermarking algorithm produces the best result and most resilient to the distortive attacks but it can be easily removed.
iii. Qu/Potkonjak static watermarking algorithm is the weakest algorithms while it does not successfully embedded any watermark.
iv. Proguard optimizer produces the best results-with a lower number of watermark recognitions for all Watermarkers, except the String Constant.

Vol 8. No. 4, December, 2015
ISSN 2006-1781

African Journal of Computing & ICT

www.ajocict.net

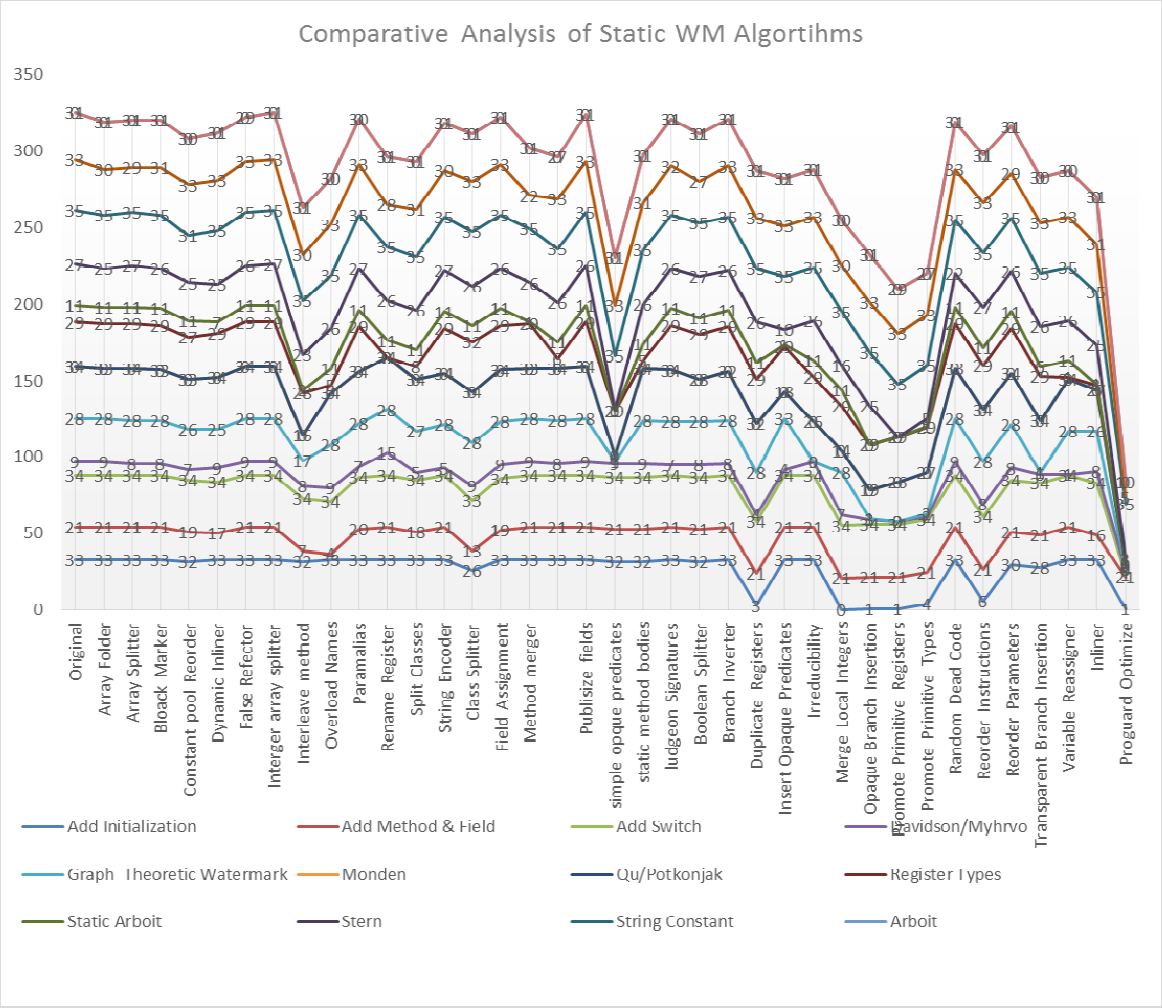

Figure 4:depicts the number of successful watermarks recognized embedded by static software watermarking algorithms.

5. CONCLUSION

Software Piracy is one of biggest problem for software industry, causing loss of millions of dollars every year to the software industry. Software Watermarking is technique which had proven good enough to battle against the software piracy. The technique not protect but helps in finding the source of illegal distribution of software and taking legal action against them. We have described an evaluation of distortive attacks against the static Java bytecode watermarking algorithms implemented within SandMark and confirmed that most watermarks embedded by these algorithms not much resilient to the distortive attacks applied by obfuscation algorithms.

From the above results we can conclude String constant watermarking algorithm produces best results but it can be easily removed. Software watermarking must be incorporated with other form of protection such as obfuscation [2,13] or tamper-proofing techniques [6] in order to better protect software from copyright infringement and decompilation.

6. FUTURE WORK

Future work involve extending the evaluation of static algorithms with subtractive and additive attacks to find resilience of these algorithms against these attacks. We would like to use some large programs in our test cases. We will use program slicing techniques [9] in order to perform subtractive water attacks.

REFERENCES

[1] http://globalstudy.bsa.org/2013/index.html Last accessed 20/5/2015

[2] C. Collberg and J. Nagra, "Surreptitious Software: Obfuscation, Watermarking, and Tamper proofing for Software Protection", Addison Wesley Professional, 2009.

[3] L. Ertaul and S. Venkatesh, "Novel obfuscation algorithms for software security," in 2005 International Conference on Software Engineering Research and Practice, SERP'05, june 2005, pp. 209–215.

[4] L. Ertaul and S. Venkatesh, "Jhide - a tool kit for code obfuscation," in 8th IASTED International Conference on Software Engineering and Applications (SEA 2004), Nov. 2004, pp. 133–138.

[5] K.Krishan, P Kaur"A Generalized Process of Reverse Engineering in Software Protection & Security", *IJCSMC, Vol. 4, Issue. 5, May 2015, pg.534 – 544,,* ISSN 2320–088X.

[6] G. Myles, "Using software watermarking to discourage piracy," Crossroads - The ACM Student Magazine, 2004. [Online] Available: http://www.acm.org/crossroads/xrds10-3/ watermarking.html

[7] G. Myles and C. Collberg, "Software watermarking via opaque predicates: Implementation, analysis, and attacks," in ICECR-7, 2004.

[8] J. Sogiros, "Is protection software needed watermarking versus software security," http://bb-articles.com/watermarkingversus- software-security, Mar. 2010. [Online]. Available: http://bb-articles.com/watermarking-versus-software-security.

[9] M. Weiser, "Program slicing," in ICSE '81: Proceedings of the 5th international conference on Software engineering. Piscataway, NJ, USA: IEEE Press, 1981, p. 439449.

[10] C. Collberg, "Sandmark algorithms," University of Arizona, Department of Computer Science, Tech. Rep., Jul. 2002.

[11] C. Collberg and C. Thomborson, "Software watermarking: Models and dynamic embeddings," in Proceedings of Symposium on Principles of Programming Languages, POPL'99, 1999, pp. 311–324.

[12] C. Collberg, C. Thomborson, and D. Low, "On the limits of software watermarking," in Technical Report #164, Department of Computer Science, The University of Auckland, 1998.

[13] K. Kumar, P. Kaur "A Thorough Investigation of Code Obfuscation Techniques for Software Protection", IJCSE, Vol.-3(5), PP:158-164 May 2015.

[14] W. Zhu, C. Thomborson, and F.-Y. Wang, "A survey of software watermarking," in IEEE ISI 2005, ser. LNCS, vol. 3495, May 2005, pp. 454–458.

[15] Christian S. Collberg and Clark Thombor-son. Watermarking, tamper-proofing, and obfuscation - tools for software protection.In IEEE Transactions on Software Engineering, volume 28, pages 735–746, August 2002.

[16] M.R. Stytz, J. A. Whittaker, "Software Protection-Security's Last Stand", *IEEE Security and Privacy,* January/February 2003, pp. 95-98.

[17] J. Nagra and C. Thomborson, "Threading software watermarks," in IH'04, 2004.

[18] G. Qu and M. Potkonjak. *Analysis of Watermarking Techniques for Graph Coloring Problem*, Proceding of 1998 IEEE/ACM International Conference on Computer Aided Design, ACM Press, 1998, pp. 190-193.

[19] G. Qu and M. Potkonjak, "Hiding signatures in graph coloring solutions," in Information Hiding, 1999, pp. 348–367, citeseer.nj.nec.com/308178.html.

[20] world-wide developer team, "jEdit - programmer's text editor," 2015. [Online]. Available: http://www.jedit.org/

[21] G. Arboit, "A method for watermarking java programs via opaque predicates," in The Fifth International Conference on Electronic Commerce Research (ICECR-5), 2002. [Online]. Available: http://citeseer.nj.nec.com/arboit02method.html.

[22] [22] C. Collberg and T. R. Sahoo, "Software watermarking in the frequency domain: implementation, analysis, and attacks," J.Comput. Secur., vol. 13, no. 5, pp. 721–755, 2005.

[23] J. Nagra, C. Thomborson, and C. Collberg, "A functional taxonomy for software watermarking," in Aust. Comput. Sci. Commun., M. J. Oudshoorn, Ed. Melbourne, Australia: ACS, 2002, pp. 177–186.

[24] J. Hamilton and S. Danicic, "An Evaluation of the Resilience of Static Java Bytecode Watermarks Against Distortive Attacks", Int. J. of Computer Science, International Association of Engineers (IAENG), HongCong, vol. 38, no. 1, pp. 1-15, 2011.

[25]http://proguard.sourceforge.net/

[26] K. Krishan, P.Kaur, "A Comparative Analysis of Static and Dynamic Java Bytecode watermarking Algorithms" Accepted at Computer society of India 2015 Conference.

Authors Biography

Krishan Kumar received his B. Tech. in Computer Science & Engineering from Shaheed Bhagat Singh College Of Engg. & Technology Ferozepur and Completed M. Tech. (Software Systems) from Guru Nanak Dev University, Amritsar. He is working as Assistant Professor in Department of Science and Technology, ICFAI University, Baddi, HP, INDIA. His area of Research is Software Protection, Reverse engineering, Malware analysis and Soft Computing.

Prabhpreet Kaur is working as an Assistant Professor in the department of Computer Engineering & technology, Guru Nanak Dev University, Amritsar. Her area of research is Digital Image Processing and Software Engineering.

Viney Kehar is working as an Assistant Professor in Department of Computer Science & Engineering, National Institute of technology, Hamirpur(HP). His area of research is Wireless Sensor Network and Software Engineering.