International Journal of Security and Its Applications
Vol. 10, No. 7 (2016) pp.147-156
http://dx.doi.org/10.14257/ijsia.2016.10.7.13

An Evaluation of Dynamic Java Bytecode Software Watermarking Algorithms

Krishan Kumar[1], [2]Viney Kehar and Prabhpreet Kaur[2]

{[1]Krishankumar@iuhimachal.edu.in}, {[2]Vineykehar@gmail.com}
{[3]prabhsince1985@yahoo.co.in}
[1]Faculty of Science and Technology, ICFAI University, Baddi(HP)
[2]Department of Computer Sci. & engineering, NIT, Hamirpur
[3] Department of Computer Science and Technology, Guru Nanak Dev University, Amritsar

Abstract

In the era of Information technology, Software Piracy and security has become one of the most important issue in world. Numbers of techniques have been proposed and implemented to prevent software piracy and illegal modification. Among all the protection techniques, software watermarking technique 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 evaluate the existing dynamic watermarking algorithms using them to watermark java bytecode files and then applying distortive attacks to each watermarked program by obfuscating. Our study has shown that some watermarks were removed as results of these transformations.

***Keywords**: Software Piracy, Java Bytecode, Watermarking, Software Protection, Reverse engineering, obfuscation*

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 techniques 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 are going to perform an evaluation of dynamic software Java bytecode watermarking algorithms from SandMark [10] framework ie. CT and Dynamic Arboit Algorithm. 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. 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.

2. Background

Watermarking algorithms are utilized widely in the multimedia industry to recognize the multimedia files such as video and audio files, and similar idea has extended to software industry. The motivation behind 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 related with implementation of software watermarks and some of the watermarking techniques are vulnerable against various attacks such as distortive attacks. Watermarking software must meet the following set of constraints:

1. Program size: embedded watermarks should not increase the size of program significantly.

2. Program efficiency: effectiveness of watermarked system or software must be like unique program and need not be diminished essentially

3. Robust watermarks: Embedded watermarks must be sufficiently solid to distortive or semantics saving changes

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 troublesome issue which is have to unravel is keeping the watermark avoided enemies while in the meantime, permitting the product proprietor to proficiently extricate the implanted watermark when required. In the event that it sufficiently simple then a foe would have the capacity to concentrate watermark as well. In the event that the watermark is concealed well then programming proprietor may have issue in separating the watermark.

2.2. Watermarking Techniques

Software watermarks can be classified into two classifications: static and dynamic [11]. Static watermarks strategies embeds the watermark in the information/or code of the project while dynamic watermarking systems inserts the watermark in an information structure assembled at run-time.

Static watermarks are installed in the information and/or code of a system. For instance, install a copyright notice into the strings. If there should be an occurrence of Java projects, watermarks could be installed inside of their consistent pool or system groups of java class records.

As before the scholastic exploration in the field of software watermarking began, some of pioneer static software watermarking systems was displayed in patents [11, 12]. The principle issue with embedding a string watermark in software is that pointless variables could likewise be effectively uprooted by performing dead-code examination, and the majority of the times when obfuscation or optimization of code is applied numerous useless method or variable names are either lost or renamed.

2.3. Classification of Watermark

Nagra *et. al.*, classified the watermark into four types [2, 14]:

Authorship Marks are utilized to recognizing a product creator, or creators. These watermarks are basically unmistakable and hearty to the assaults.

Fingerprinting Marks are utilized to serialize the spread article by implanting an alternate watermark in each circulated duplicate. It is utilized to discover the system or channel of dissemination, *i.e.*, the individual who has unlawfully circulated the duplicates of software. The watermarks are for the most part strong, undetectable and comprise of a one of a kind identifier *e.g.*, client reference number.

Validation Marks are utilized by basically end clients to confirm that product item is authentic, real and unaltered, for instance if there should arise an occurrence of Java, digitally marked Java Applets. A typical system is to process the summary of programming item and implants into programming as a watermark. An overview is registered by utilizing the MD5 or SHA-1. An acceptance imprint ought to be delicate and obvious.

Licensing Marks are utilized to guarantee the product is valid against a permit key. One property of these watermarks are that they are fragile .The key ought to wind up futile if the watermark is harmed.

2.4. Types of Attacks to Watermarks [2]

I. Distortive attacks: these kinds of attacks includes applying the semantics preserving changes to a product, for example, improvements or confusions, in this way uprooting any watermark which depend in system language structure.

II. Additive attack: In these attacks, another watermark is added by an adversary to the effectively watermarked system so as to provide reason to feel ambiguous about which watermark was included first [7].

III. Subtractive attacks: In these attacks, an aggressor decompiled or dismantled the code keeping in mind the end goal to expel the watermark from the project.

2.5. Dynamic Watermarking

Dynamic watermarking techniques embeds a watermark in a program's execution state which is built at runtime *e.g.*, in control flow graph (CFG) rather than embedding into the program code itself [11, 2]. Therefore dynamic watermarking techniques should be resilient to semantics preserving transformations which are applied by obfuscators or code optimizations.

3. The Empirical Evaluation

We are going to evaluate Dynamic watermarking software techniques by watermarking the 35 [20] jarfiles with the available watermark algorithms and then apply a distortive attack to each watermarked program, by obfuscation. After all the programs have been transformed we attempt to extract the watermarks from the obfuscated programs. We expect 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.

The Watermarker

We are going to test the two dynamic watermarking algorithms from 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.

A. Dynamic Watermarking Algorithms are as:

I. Arboit algorithm [21]: in this a trace of the program is used to select the branches. The watermark can be encoded in the opaque predicate by ranking the predicates in the library and then assigning each predicate a value or by using constants in the predicated to encode. It is also possible to embed the watermark through the use of opaque methods. In this case a method call is appended to the branch and this method evaluates the opaque predicate. If the watermark is encoded using the rank of the predicate then it is possible to reuse the opaque methods to further distinguish the watermark.

II. TheCollberg-Thomson Algorithms [10]: This algorithm is a dynamic software watermarking method that embeds the watermark in the topology of a graph structure built at runtime. Watermarking a Java jar-file using the CT algorithm and recognizing that watermark requires several phases. First, the source program has to be **annotated**. This means that calls to sandmark.watermark.trace.Annotator.mark() are added to the source program in locations where it is OK to insert watermarking code. Next, the source program is compiled and packaged into a jar-file. Then the program is **traced**, *i.e.*, run with a special (secret) input sequence. This constructs a trace-file, a sequence of mark()-calls that were encountered during the tracing run. The next step is **embedding** which is where the watermark is actually added to the program, using the data from the tracing run.

4. The Results

A dynamic algorithm relies on information gathered from the execution of the application to embed and recognize the watermark. Numbers of static watermarking algorithms available are more in number as compare to dynamic algorithms due to the multitude of locations where information can be hidden in an executable. Out of three Dynamic watermarking algorithms available in Sandmark, two algorithms CT algorithm and Dynamic Arboit algorithm are tested and resulted are as shown below:

4.1. Watermarking

After embedding watermark we have obtained 65 out of an expected 70 watermarked jars. 5 watermarked jar files failed to embeds the specified watermarks, due to error or incompatible program jar.

Table 1. Show the Percentage of Successful Embedding of Dynamic Watermark

	Total	Successful	Failed
Watermarks embeds	70	65	5
%age		92.85	7.15

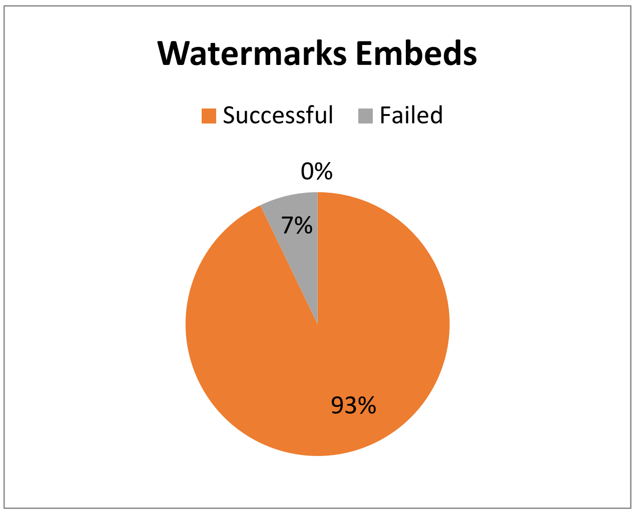

Figure 1. Depicts that around 93% Successfully Watermarked Jar Files are Obtained after Watermarking

Out of the 65 watermarked jar files only 62 contained watermarks which were successfully recognized before the transformations attacks were applied. This means success rate is 95.38 of the expected watermarked jar files produced actually recognized.

Table 2. Shows the Percentage of Watermarks Recognized after Watermarking

	Total	Successful	Failed
Watermarks embeds	65	62	3
%age		95.38	4.62

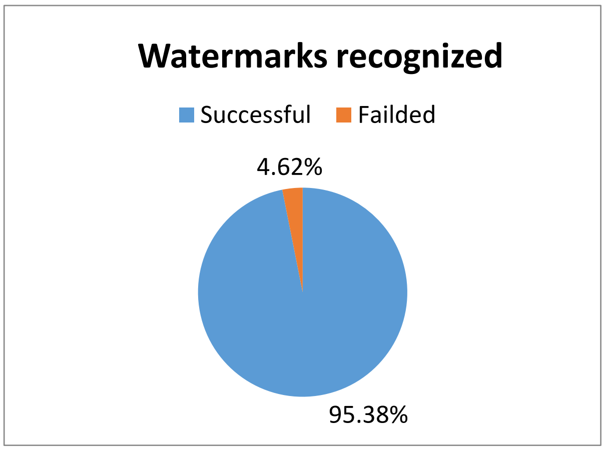

Figure 2. Depicts that More than 95% Embedded Watermarks are Recognized before Applying Obfuscation

4.2. Obfuscation

We obfuscated the 65 jar files with 36 obfuscation, which would have resulted in 2340 attacked watermarked jars. There are some algorithms failed to output some jars so we actually obtained 2253 attacked watermarked jars using 36 semantics preserving transformations.

Table 3. Show the Percentage of Obfuscated Jar Files after Applying Transformation with 36 Obfuscations Algorithms Watermarked by CT Algorithm

	Total	Successful	Failed
CT Algorithm	1188	1135	53
%age		95.54	4.46

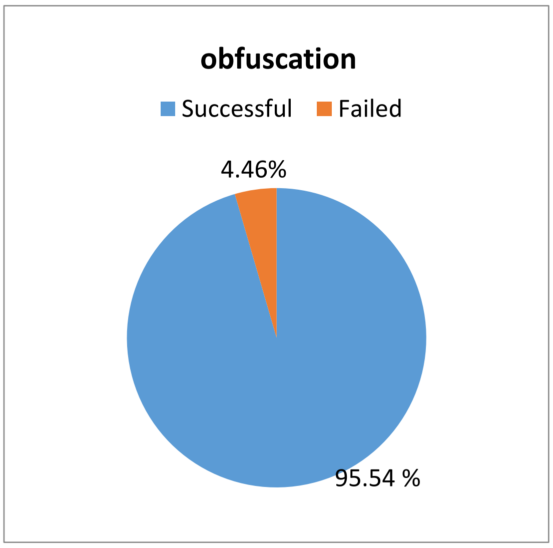

Figure 3. Depicts that more than 95 % Obfuscated Jar Files are Obtained after Applying Obfuscation in case of CT Algorithm

In case of Dynamic Arboit algorithm results are shown below:

Table 4. Show the percentage of obfuscated jar files obtained after applying transformation with 36 obfuscations algorithms watermarked by Dynamic Arboit Algorithm

	Total	Successful	Failed
Dynamic Arboit	1147	1118	29
%age		97.47	2.53

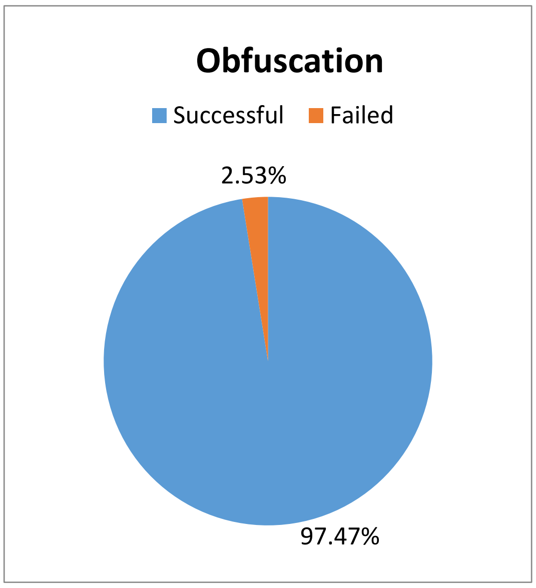

Figure 4. Depicts that more than 97 % obfuscated Jar files are obtained after applying obfuscation in case of Dynamic Arboit algorithm

4.3. Recognition

Result of recognizing the embedded watermark in the obfuscated jar files are shown by line graph in case of Dynamic CT and Arboit watermarking algorithm. Figure 5 below show the bar line marked with numbers which show the number of jar files successfully recognized the watermark obfuscated by respective obfuscation algorithms shown on x-axis.

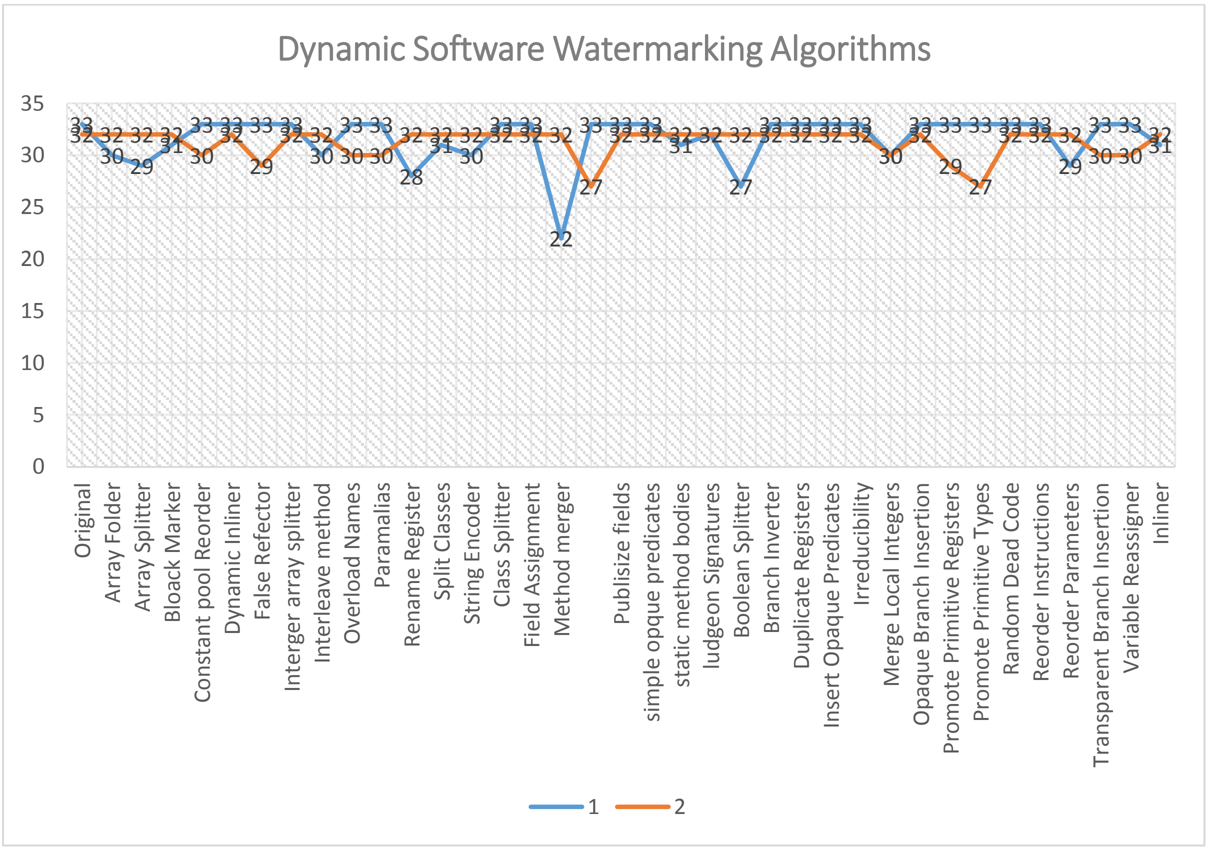

Figure 5. Depicts the number of successful watermarks gets recognized, embedded by CT and Arboit watermarking algorithm against the obfuscation algorithms

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 dynamic Java bytecode watermarking algorithms such as CT Algortihm and dynamic Arboit algorithm and confirmed that most watermarks embedded by these two algorithms such as CT and dynamic Arboit are resilient to the distortive attacks applied by obfuscation algorithms. From the above results we can conclude that CT algorithm is slightly more resilient than Dynamic Arboit.

In future we will try to extend our evaluation using the slicing technique to extract embedded watermark.

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, **(2005)** June, 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), **(2004)** Nov., pp. 133–138.

[5] K.Krishan, P Kaur, “A Generalized Process of Reverse Engineering in Software Protection & Security”, IJCSMC, ISSN 2320–088X, vol. 4, Issue. 5, **(2015)** May, pp. 534 – 544.
[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, **(2010)** Mar. [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)**, pp. 43.
[10] C. Collberg, “Sandmark algorithms”, University of Arizona, Department of Computer Science, Tech. Rep., **(2002)** Jul.
[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 and P. Kaur, “A Thorough Investigation of Code Obfuscation Techniques for Software Protection”, IJCSE, vol. 3, no. 5, **(2015)** May 4, pp. 158-164.
[14] W. Zhu, C. Thomborson and F.-Y. Wang, “A survey of software watermarking”, in IEEE ISI 2005, ser. LNCS, vol. 3495, **(2005)** May, pp. 454–458.
[15] C. S. Collberg and C. Thombor-son, “Watermarking, tamper-proofing, and obfuscation - tools for software protection”, In IEEE Transactions on Software Engineering, vol. 28, **(2002)** August, pp. 735–746.
[16] M.R. Stytz and J. A. Whittaker, “Software Protection-Security’s Last Stand”, *IEEE Security and Privacy*, **(2003)** January/February, 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*, Proceeding 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 coloringsolutions”, 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] C. Collberg and T. R. Sahoo, “Software watermarking in the frequency domain: implementation, analysis, and attacks,” J.Comput. Secur., vol. 13, no. 5, **(2005)**, pp. 721–755,
[23] W. F. Zhu, “Concepts and techniques in software watermarking and obfuscation”, PhD Thesis, The University of Auckland, **(2007)**.

Authors

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 Faculty Member at ICFAI University, Baddi, Himachal Pradesh. His area of Research is Software Protection, Reverse engineering, Malware analysis and Soft Computing.

Prabhpreet Kaur, is working as 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 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.