An Improved Secured Client Authentication to Protect Software against Piracy

G. Syam Prasad
B.Tech,M.Tech, Ph.D
Associate prof &HOD in CSE Dept
Usha Rama college of Engineering
& Technology, Telaprolu,
Vijayawada.

G.Samuel Vara Prasad Raju
M.Tech., Ph.D
Professor in CSE Department
Andhra University, Visakhapatnam

ABSTRACT
Software protection and security plays a vital role to the business and commercial organizations. Various techniques has been proposed to enhance the security of the client's system to authenticate the software against piracy protection. Various software attacks like hardware cloning, software cloning ,software cracking, virus scripts provides high influence on economic development. So it is necessary to develop an enhanced framework to protect software systems against piracy. Existing approaches provide solutions against using smart cards or internet based secured user authentication mechanisms. Existing literature work mainly provides solutions to software piracy against different attacks. But existing approaches doesn't protect software's through reverse engineering or hardware cloning threats. This paper proposed a framework which can be used to protect piracy against hardware piracy. Experimental results shows proposed approach provides better software protection against existing smart card based techniques.

Keywords
Software Protection, Authentication, Hashing, Internet, Piracy.

1. INTRODUCTION
Despite the high public profile of piracy as a threat to intellectual property owners, surprisingly little useful research continues to be done to understand the variety technical solutions that may feasible. Piracy, like credit card fraud and computer security, hazards go which cannot be solved completely.

Research identified technical systems that offer content owners the ability to control their risk. The most practical and effective of these combine programmable code with encrypted digital content. This code could be distributed along with content, execute dynamically during playback, and enforce each title's security policies. Publishers could then control security for his or her own content. Various robust technologies and smart cards , have been applied as solutions to the problem of piracy, but most of the commercial smart card devices failed once an implementation is compromised.

Software-splitting [1] is naturally a conceptually simple as well as appealing technique for protecting software from piracy. Remove small but essential components that are caused by the application putting them on a secure server, either on any secure coprocessor or across the Internet. The server provides the missing functionality, but rarely the missing components. If reverse engineering the constituents that are caused by the functionality is very tough, the server may have absolute control over the circumstances under which is situation software may be used.

Software cracking serves as a serious threat to several within the software industry. It s the problem in which a cracker, having obtained a copy of a given software he likes to attack, succeeds in breaking the protection that typically comes built into it. Typically, crackers would create modified little examples of the software, or crackz, whose copy protection or usage control mechanisms have been disabled. Cracked software can then be illegally redistributed to the public, exacerbating the application piracy problem. With commerce and distribution of copyrighted multi-media rapidly moving online, the demand for software protection is more urgent than before: client software code running on untrusted machines really needs to be secured against tampering[2].

Protection mechanisms that will effectively protect software running in untrusted environments should have some fundamental properties:

Resilience: The protection does not have any single points of failure that is more difficult to disable.

Self-defense: Ready to detect and immediately take actions against tampering (i.e., code modification).

Checksum another section of program code at runtime and verify its integrity (i.e., check if it was tampered with). In the event the guarded code can be found altered, the guard will trigger whichever sequence of actions is desired regarding the situation, directly from the mildest of silently logging the detection event, into the extreme of producing the software program un- usable (e.g., by halting its execution, or even better, causing an eventual crash that is going to be hard to trace back to the guard). If no code changes are detected, the program execution proceeds normally. Programs guarded by checksumming guards are made, using some sense, \self-aware\" of their own honesty.

There are actually three major threats recognized contrary to the intellectual property contained in software. Software piracy happens to be the illegal reselling of legally obtained copies regarding a program. Software tampering happens to be the illegal modification of a program to circumvent license checks, must purchase use of digital media protected through software, etc. Malicious reverse engineering would be the extracting of causing section of a program as a way to reuse it in ones own[9].

Protecting code from attacks such as reverse engineering , analysis and tampering attacks is among the main concerns for software providers. If a competitor succeeds in obtaining and reusing a algorithm, it'd bring about major issue. Moreover, secret keys, confidential data or security related code typically are not planned to be examined, extracted,

stolen or corrupted. No matter if legal actions which can include patenting and cyber crime laws are in place, these techniques remain an important threat to software developers and security expert[3].

This paper provides a solution on software protection and gives better client server secure software installation process which would encourage further research to protect software against piracy. This paper additionally provides various viewpoints, discuss challenges and suggest future directions.

2. BACKGROUND AND RELATED WORK

Collberg C.S. and Thomborson C. et al. [3] provided a compact outline approaches to protect against these threats. Piracy, reverse engineering and tampering have already been the key software threats. Software watermarking which specializes protecting software reactively against piracy. When that data is distinctive for each example, anyone can mark out copied software into the source unless the watermark is smashed. It usually implants hidden, distinctive data into a remedy in such a manner that it may be guaranteed that a certain software instance is one amongst a certain individual or company. The next group, code obfuscation, protects the software from reverse engineering attacks. This approach comprises of one of these program alterations that alter a program in such a manner that its functionality remains identical but analyzing the internals of the program becomes very tough. Another team of approaches focuses in order to make software "tamper-proof", referred to tamper-resistant. Most of the developed software products which overcomes the reverse engineering attacks while developing the code segments[4-6].

Most software tempering is completed not by altering a lot of the executable.[4]Oblivious hashing requires provide both results and stealth. Attention is paid to blend executable code with hashing code, making it difficult for hackers to extract to get program that executes the program. Wile program is compiling with the degree of generating the tree of execution, hashing code is injected and hence shares the very same registers and executes the same manner like the application making it virtually impossible to identify. Oblivious hashing computes an incredible hash value dependent upon the order of execution, therefore is aware of the the process software executes.

In this particular paper, Hongxia Jin et al., [5] concentrates on the attacker identification and forensic examination. This paper focuses on the protection associated with a software application and the content, many have seen billions of dollars spent each year through industries just for software piracy and digital media piracy. The achievement of the content/software security within the huge segment is based on the flexibility of protecting software code against tampering and identifying the attackers who issue the pirate copies. This paper discussed in regards to proactive detection approach for defeating an on-going attack prior to a cooperation has occurred. The author also describes another detection approach for post-compromise attacker identification. But this system has limitation regarding the user identification against the hardware piracy.

In [6] proposed a secure user authentication mechanism between client and server using Elliptical cryptography algorithms. Software authors can use strong encryption to delay the disassembly of their own applications. Lacking the proper decryption key or algorithm, the encryption defeats both the static and dynamic analyses. An attack must either defeat the encryption process itself or find yet another way to search for the decryption key. After obtaining the important thing, the attacker can decrypt the encrypted binary revealing the binary executable. This case is perfect for reverse-engineering, because of the fact that the reverser is capable of doing static and convincing analysis upon the deciphered application. Often times, the reverser can dynamically analyze this program, because many programs decrypt themselves during execution. The decryption method can use an internally or externally stored key. A good developer can store the decryption key in the program possibly in an encoded form or calculate it at runtime. Then again, the developer could store the important thing external to this program either on a local hardware device or on an overseas key server. Among the latter scenario, the program requests the key from the key server at runtime and of course the server would only supply the encryption key after authenticating the client[7,8].

3. PROPOSED SYSTEM

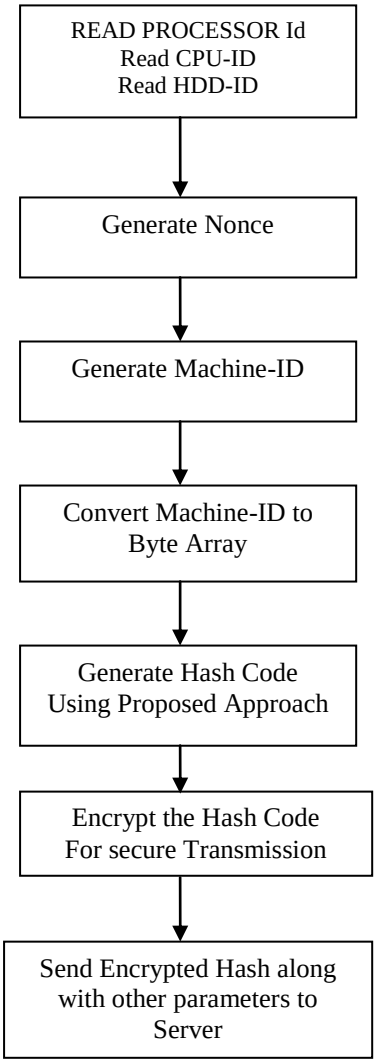
Software Piracy protection in this proposed framework is achieved in two phases as Client phase and Server Phase. In the client phase user system details are requested and stored securely in the software product manufacturers company. In the second phase user's system details are validated at the company's server in order to check whether the user's license is standalone or not. The whole working process in each phase is described in this section.

3.1 Client Side Registration Phase

Before deploying the software product, the software manufacturer company has to embed proposed secure privacy mechanism in their product for software protection and user validation. After delivering the product all the users has to register the product through this secure client registration phase.

In the client registration phase all the users has to install the product with internet connectivity. The overview architecture of client side process is given below:

3.2 Client Side Registration Process



In this process client system reads processor_id,cpu_id and hdd_id from the client system along with nonce random number. After reading security parameters from the system machine_id is generated as:

MachineID Generation:

Machine_ID=Concat(Processor_id,Cpu_id,Hdd_id)mod256.
If ((Bytes lengths of Machine_ID) ==256)
Flag=true;
Else
PaddingSize=256-Size(Machine_ID);
Machine_ID=Machine_ID+Padding_bytes;

Whirlpool Hash Function:

The Whirlpool Hash Algorithm is 512-bit hash function designed by Vincent Rijmen and Paulo S.L.M. Barreto is one of the best hashing approach used for privacy

Whirlpool is block cipher based hash function intended to provide security and performance that is comparable than that found in non block cipher based hash functions such as SHA. Whirlpool has the following features:

- Hash code is 512 bits.
- The overall structure of the hash function is one that has been shown to be resistant to the usual attacks on block cipher based hash codes.
- Uses Static S box.
- Easy to attack intermediate matrices using existing shifting columns operations.

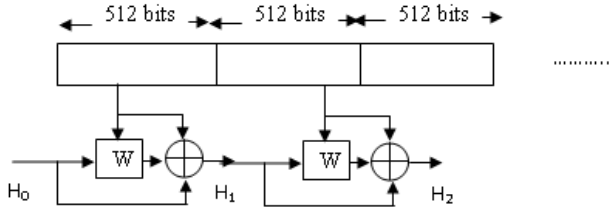
Proposed Hash Function:

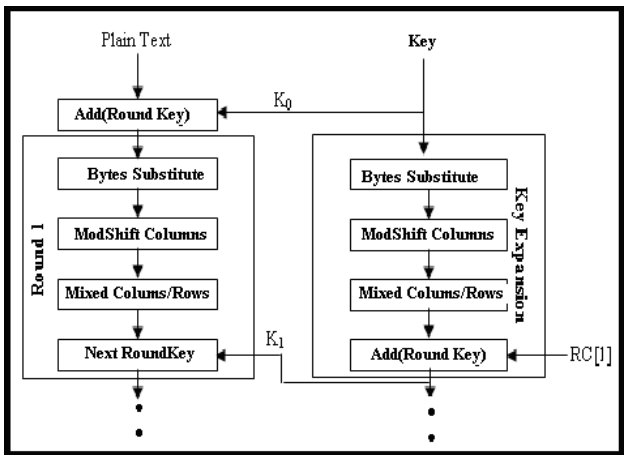
Hash Function Structure:

Given a message consisting of a sequence of blocks b1;b2;…bt, the Modified hash function is expressed as follows:

HASH(0) = IV
HASHi = W(HASH(i-1);bi) +(HASH)i-1 + bi
HASHt = Final Hash Code.

Proposed algorithm takes maximum length of 512 bits as input a message and produces as output a 512-bit message digest.



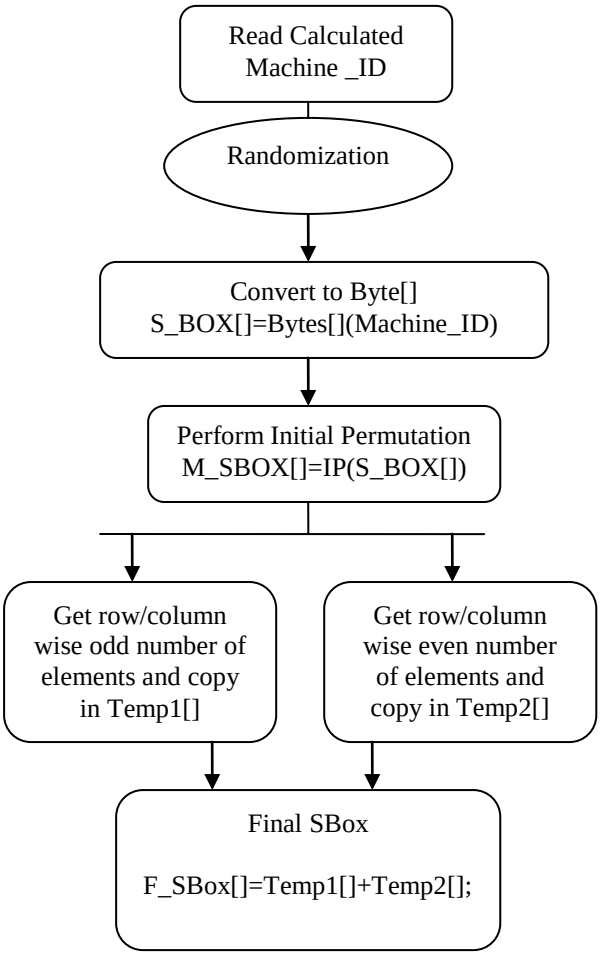


For each Round following Four operations are performed :

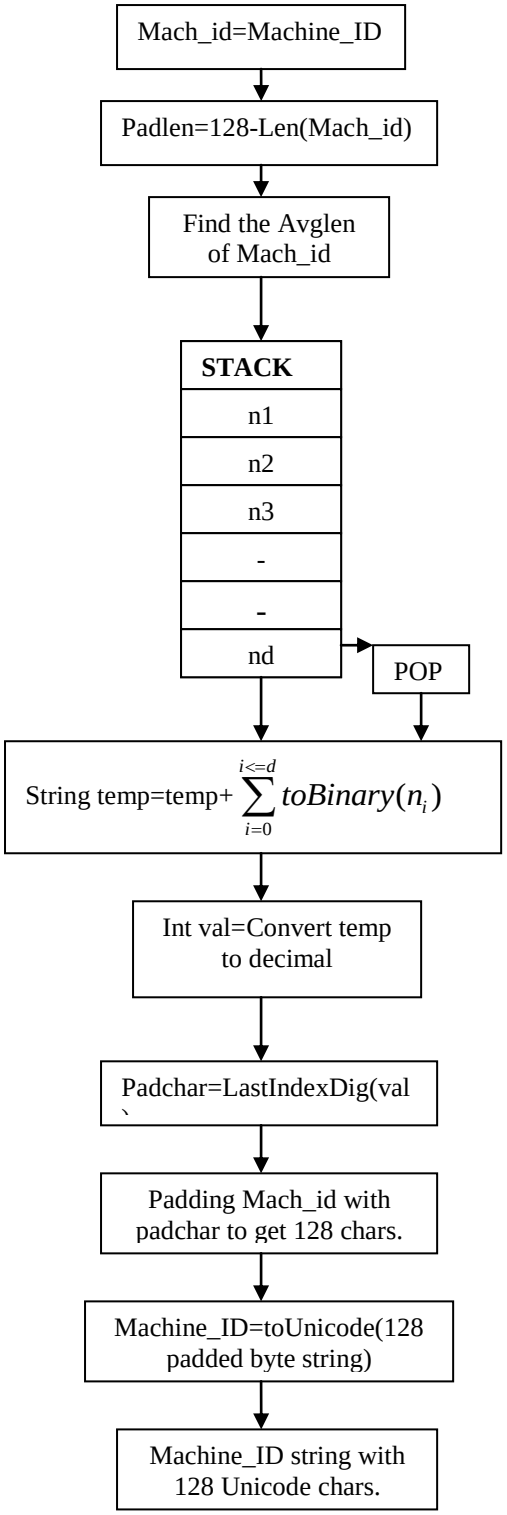
- Add Key in Each Round and Next Round.
- Bytes Substitution Approach.
- Shift columns using modified approach.
- Mix Rows/Columns for security.

This process is repeated until all rounds are executed. Finally Hash code is generated.

Dynamic S-BOX



Randomization:



In the Dynamic S_BOX for each client it will generate unique S_BOX dynamically. Based on the Machine_ID it will generate unique Dynamic SBOX.

First it reads clients Machine_ID and then it converts into byte[] as S_BOX[]. After that S_BOX[] is initially Permuatated it will save into M_SBOX[]. After the M_SBOX[] is initialized with initial Permutation,

i) Get row/column wise odd elements and copy into temp1[].

ii) Get row/column wise Even elements and copy into temp2[].

Finally SBOX is generated by permuting the concatenation of both temp1[] and temp2[].

Example:
Proposed algorithm generates 256 size dynamic sbox Unicode value is

Machine id:
75J66BSBFEBFBFF000206522020202020202020202020205636374538585143

SBOX value:
\u0037\u0035\u004a\u0036\u0036\u0042\u0053\u0042\u0046\u0045\u0042\u0046\u0042\u0046\u0046\u0030\u0030\u0030\u0032\u0030\u0036\u0035\u0032\u0032\u0030\u0032\u0030\u0032\u0030\u0032\u0030\u0032\u0030\u0032\u0030\u0032\u0030\u0032\u0030\u0032\u0030\u0032\u0030\u0032\u0030\u0032\u0030\u0035\u0036\u0033\u0036\u0033\u0037\u0034\u0035\u0033\u0038\u0035\u0038\u0035\u0031\u0034\u0033\u0034\u0034

Creating a dynamic SBOX is based on client system condition as:

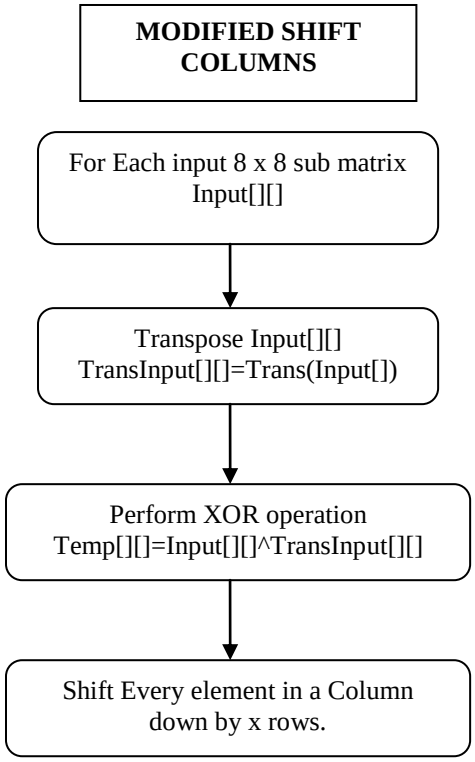
```
If( Machine_ID exist in ServerDB)
{
Use Existing SBOX;
}
Else
{
Create new SBOX;
}
```

Initial Permutation:

```
i = 0;
for j = 0 to 255;
do i = (i + SBOX[j])mod256;
swaping (SBOX[j] , SBOX[i]);
end;
```

MODIFIED SHIFT COLUMNS:
The Shift Every element in a Column down by x rows where x equals to the column number cause a circular downward shift of each column of state c except the first column. For the second column, a 1-byte circular downward shift is performed; for the third column, a 2- byte circular downward shift is performed; and so on.

For each sub matrix of input size 8x8 transpose the matrix i.e each rows is transform to corresponding columns. After transformation is performed XOR operation is performed to initial sub matrix and transposed matrix .



3.3 Server Side Validation Phase

Company Server receives the client's system information in encrypted format. Decrypts the encrypted information and then verifies the client's hardware information to protect against piracy.Server side validation phase has following steps:

1) Decrypt the E(3DES,Nonce,HCODE,security parameters,ipaddress,others);
2) Using security parameters it will calculate Machine_ID.
3) Checks whether the Machine_ID is exist or not.
4) If Machine_ID exists then it will returns Already registered message.
5) If Machine_ID not exist in server DB then the client machine will returns activation code and then installation process will starts at the client's machine.
6) After successful installation of the software activation code is saved in the company server DB.

4. EXPERIMENTAL RESULTS

All experiments were performed with the configurations Intel(R) Core(TM)2 CPU 2.13GHz, 2 GB RAM, and the operating system platform is Microsoft Windows XP Professional (SP2). This framework implementation requires hardware and internet connectivity.

Figure1: Client side HomeScreen of Proposed Work

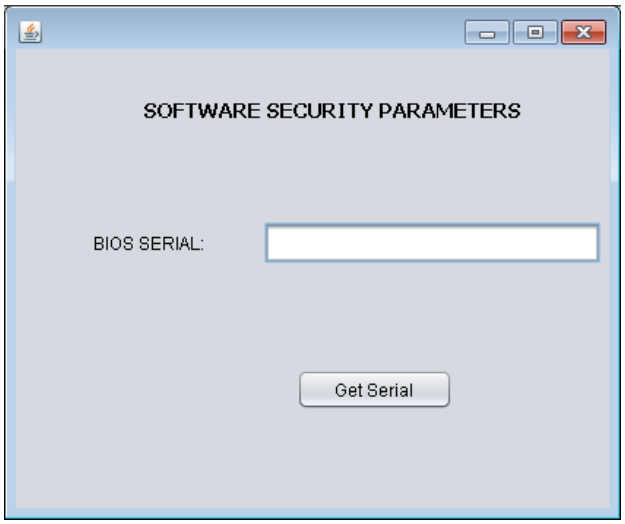


Figure 2: Get client system Bios Serial

Figure 3: Get client system cpu serial id

Figure 4: Get client system HDD serial id

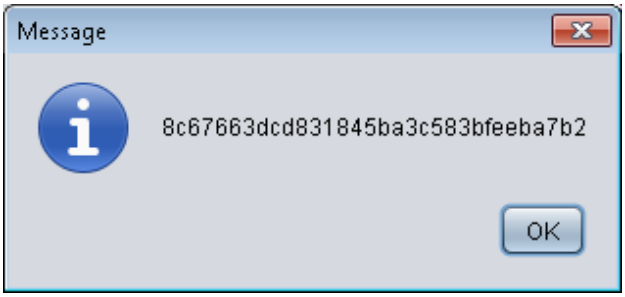


Figure 5: Generates Machine-ID

Figure 6: Client side software installation

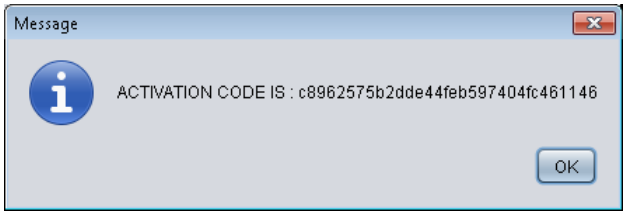


Figure 7: Generates Activation Code

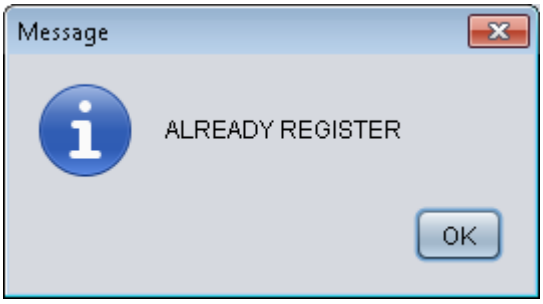


Figure 8: If client user already exists

Figure 9: Client enters activation code

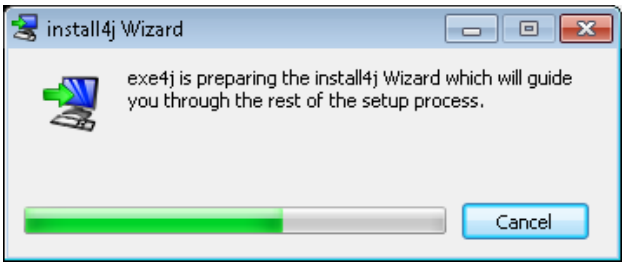


Figure 10: Client activation code is active and already registered then software starts installation.

Results Comparison:

Hardware Key:

75J66BSBFEBFBFF000206522020202020202020202020205636374538585143

MD5:
c8962575b2dde44feb597404fc461146

SHA-256:
d3a71ccadd10daf38d69f335f9763c4d68a3f51b2716404cc3299191479004e

SHA-512:
fa6c0bd08ebdf5dc5df654217d224d382066a903a4cbf3e80d80ec69a19b727d78734d558be4e3923fa04a7ef5c2a1bd84b29b005f0ca58e214d994d1905f76d.

NEW HASH:
CC5084F26D249A367F8417C218A7A5544D71887C30D7B429C08927CB4495C107EE98F1886C7F2D6CFD9A1334105CFEF37FDAF0EF8542E74A32ED21D7C779DE45

5. CONCLUSION AND FUTURE SCOPE

This system proposes a robust mechanism to protect software against piracy using two phases. First phase of client's registration provides secure registration of security parameters in the company DB. Proposed approach gives better hash value which is very difficult to break against frequency attacks. This system gives better security than existing approaches in terms of time, and security parameters are concern. Proposed Framework takes less time to generate activation code and an instance of hardware parameters for unique client registration.

6. REFERENCES

[1] Virtual Leashing: Internet-Based Software Piracy Protection Ori Dvir

[2] Protecting Software Codes By Guards Hoi Chang, Mikhail J. Atallah.

[3] Collberg C.S. and Thomborson C., "Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection", IEEE Transactions on Software Engineering, Vol. 28, Issue 8, Pp. 735 – 746, 2002.

[4] Protecting against piracy: Building Tamper resistant software By Amodha Wijekoon

[5] Hongxia Jin and Lotspiech J., "Forensic analysis for tamper resistant software", 14th International Symposium on Software Reliability Engineering, 2003.

[6] B. Pinkas and T. Sander, ―Securing passwords against dictionary at-tacks,‖ inCCS '02: Proc. 9th ACM Conf. Computer Communications Security, New York, 2002, pp. 161–170, ACM.

[7] Jan Cappaert, Bart Preneel, Bertrand Anckaert, Matias Madou, and Koen De Bosschere "Toward Tamper Resistant Code Encryption: Practice and Experience," LNCS, Vol. 4991, Pp. 86-100, 2008.

[8] Jan Cappaert, Nessim Kisserli, Dries Schellekens and Bart Preneel "Self-Encrypting Code to Protect Against Analysis and Tampering," 1st Benelux Workshop Inf. Syst. Security, 2006.

[9] A Thorough Investigation on Software Protection Techniques against Various Attacks N. Sasirekha.