Energy Efficient and Secure Offloading using Mobile Cloud Computing

Aradhana[1], Samarendra Mohan Ghosh[2], Praveen Shrivastav[3]
[1, 3]Research Scholar, Dept. of CSE. Dr. C. V. Raman University, Bilaspur (C.G.) India.
[2]Professor, Dept of CSE, Dr. C. V. Raman University, Bilaspur (C.G.) India.

ABSTRACT
Advancements in mobile computing technology have changed the user preferences. Smart mobile devices have limited memory capacity, CPU speed and battery back-up time. The mobile cloud computing provides computational offloading to migrate the intensive task from smart mobile devices to cloud. Task offloading is vulnerable to certain risks. To enhance the security on task offloading, we propose a hybrid approach that renders the request generated by mobile application and change the structure of original code of data using code mixing and dead code insertion method. The approach makes the original code harder, unreadable and difficult to reverse-engineer. Based on our analysis, the obfuscation is able to increase the complexity of the code, confidentiality and integrity during computational offloading on clouds via mobiles agent. As an outcome the proposed frame work will be able to save battery consumption and noticeable amount of execution time in a secure channel.

Keywords: Code Obfuscation, Encryption, Computation Offloading, Reverse Engineering, Mobile Agent

I INTRODUCTION

Mobile cloud computing migrates task of the process to other server that executes some events on the behalf of user and or applications .Protecting intellectual property of any code is must and is not easily achievable; unless the understanding of the intellect present in the system is unidentified. One way of securing the intellectual property is by obfuscating the system. Many types of obfuscation can be done while the functionality of the system remains same. The obfuscation provides higher level of security that makes reverse-engineering a program more difficult and economically unfeasible [1]. Other advantages of obfuscation method on distributed environment include helping to protect unauthorized access, code optimizing, hiding vulnerabilities and shrinking the size of the programme code.

II OBFUSCATION TECHNIQUES

Here are some techniques for java byte code obfuscation techniques. They can help to create a complex defence against reverse engineering and code tampering and renaming. This alters the name of class, methods and local and global variables, identifiers to make the decompiled source much harder for a human to understand.

(a) **Control Flow Obfuscation**: This method changes the sequence of execution of code and flow of data.
(b) **String Encryption**: It transforms strings using encryption and only calls their original value when required.
(c) **Structural obfuscation**: This approach convert common instructions to other form of construct potential for confusing the decompilers.
(d) **Dummy Code Insertion**: This method inserts null code inside the source code. It breaks decompilers to makes reverse-engineered code harder to analyze and extract the original form of code.
(e) **Unused Code and Metadata Removal**: This technique reduces the unreachable code and meta data of the source code for preventing the information available to an attacker.
(f) **Binary Linking/Merging**: It combines many library file and executable lie into one or more than file.
(g) **Code Mixing**:-In this approach we scrambled the code to harden the reverse engineering reverse-engineered code harder to analyze and extract the original form of code.

III PROBLEM STATEMENT

Security, privacy, integrity and trust issues exist since the evolution of mobile cloud computing. It is a big issue of trust between mobile client/user and cloud provider. The cloud provider ensures only security regarding user authentication and data security and the environment he/she is running is not malicious but its monitoring is not in clients control. Hence the necessity for developing trust models/protocols comes in consideration [2]. Data are placed in the form of various packets on different locations on cloud. This causes the security breach thus intrusions are easily affecting the environment. It is necessary to set a security protocol on the data packet. [3]. cloud providers do not provide facility of monitoring security, location monitoring, authenticity and integrity of hardware, software and data. It requires a security model to monitors the running environment [4]. In clouds user cannot identify the location of data so the issued that who controls the integrity of data on virtual machine and difficult to maintain the consistency of security and ensure audit ability of records [5]. This agent is worked like security agent by implementing trusted computing infrastructure through authenticating hardware/software integrity. [6] Author has analysed some security threats, risks and vulnerabilities associated with cloud computing. Providers do not provide facility of monitoring security, location monitoring, and authenticity.

IV BUILDING BLOCKS OF THE PROPOSED FRAMEWORK

We illustrate the main building blocks of the proposed framework using a schematic model consists of major components of each building block in Figure 1. We have implemented a mobile agent platform to provide devices.

an agent API with the proposed security mechanism for code migration and communication in the form of JAVA packages to share workload management and reduce the Battery consumption of mobile

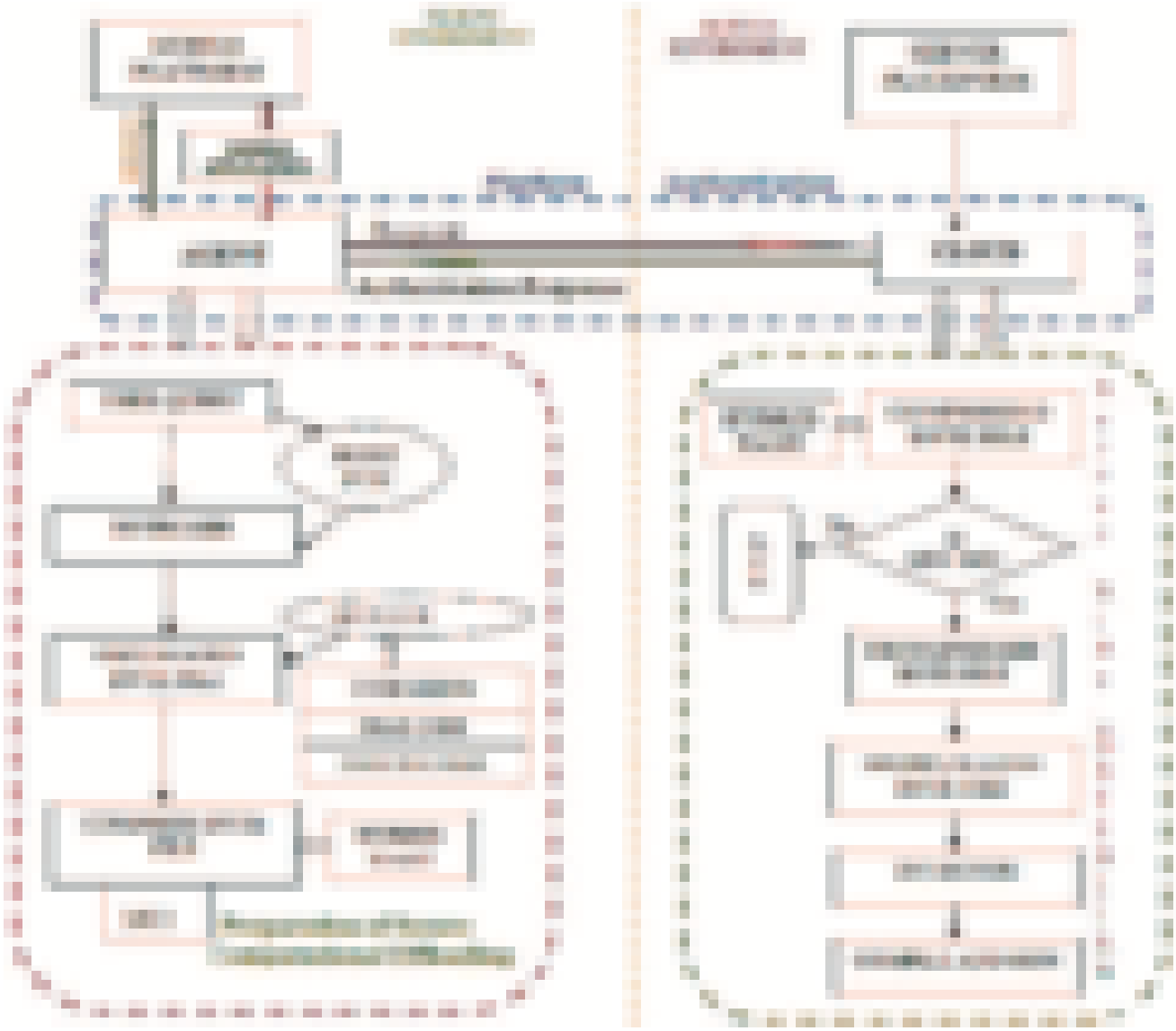

Fig. 1 : The systematic work flow of proposed Framework

V METHODOLOGY

(a) The proposed framework can be partitioned in to three parts:-

(i) Platform Authentication: **-** We have implemented an agent to protect the Client and Server Environment using JSP. The agent should have dedicated lines for individual clients. It only respond to the authenticate clients for this we apply set of protocols to give the permission for platform. The agent system often uses cookies to maintain the authentication states between agents to cloud. Every subsequent request will generate a cookie that will automatically be allowed by internet based application on agent platform. The attackers easily identify the user details using cookies. Cross site scripting (XXS) is a common attack technique that theft cookies related to subsequent request from mobile agent or web browser's databases. We proposed a new method for cookie rewriting that

encrypts the cookies and protect from Cross Site Scripting attack. This method is implemented for mobile agents it automatically rewrites the cookies therefore it protects against XXS attack.

(ii) **Code Integrity and Confidentially**:-To protect the bytecode which is off-loadable using hybrid approach of code mixing and dead code insertion techniques with cryptographic advance hash function to make our code more robust and complex and we have also applied the lossless compression technique to reduce the size of code so that it will travel faster on network.

(iii) **Sever Side Execution for Computation offloading**:- Secured code of user query will run on server environment and server will be responsible for deobfuscation process to find the original code to run and sends back the results on client environments.

VI METHOD DESIGN

In this section, we describe the methods used to enhance the security performance of this framework. To increase the security we applied the hybrid approach of obfuscation method using code mixing and dead code insertion method to make it more confidential and for the integrity security we apply cryptographic hash function that generates the message digest which detect the threat present in the transferred code.

(a) **Evolution Matrix-** In this work we have implemented Advance Cryptographic hash Algorithm with hybrid approach on obfuscation methods for code integrity and confidentially for development of secure environment with virtualization technique, malware detection and informal behavioural of monitoring. This approach makes harder for reverse engineering the code. The Framework may affect the size of program and cost of execution time.

(i) **Code Obfuscation**: - Byte code obfuscation is method of modifying the byte code or instance of executable file so that it become harder to read and understand for an attackers but it remains fully function. The methods are Renaming Control, Flow Obfuscation, String Encryption, Structural obfuscation, Dummy Code Insertion, Unused Code and Metadata, Removal Binary Linking/Merging, Code Mixing, Anti-Tamper, Class file Encryption. We apply code mixing and dead code insertion .We identified that code mixing and dead code insertion methods are the best obfuscation for byte code.

(ii) **Code Mixing**: In this approach we scrambled the code to harden the reverse engineering. In this approach we partition the code in to equal chunks and randomize the part of code with a complex manner to harden the identification of the original code.

The Pseudo code to for Code Mixing Algorithm

- Original bytecode are divide into equal chunks of SIZE=COUNT
- Chunks array are chunks[i] =1 to COUNT
- IF the code division original bytecode % SIZE ! = 0 then
- Drop the last chunk i.e. SIZE=SIZE-1
- End IF
- Circularly rotate the divided parts up to last chunks
- chunk[i]=chunk[i+2]
- End

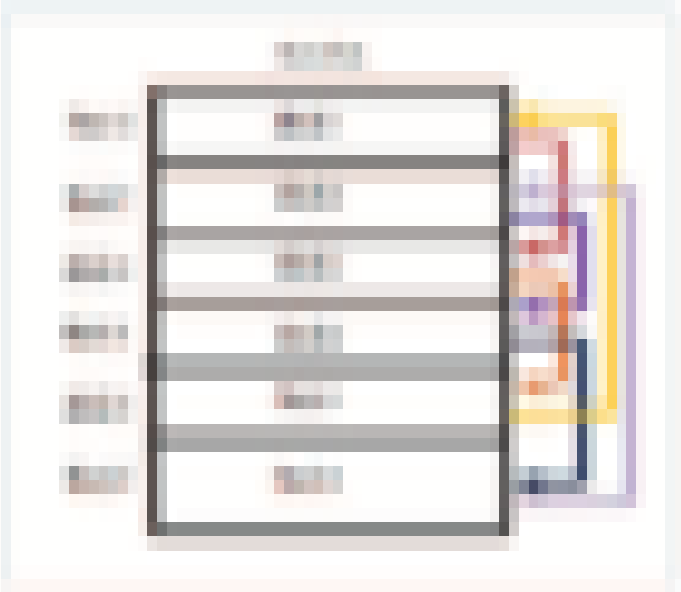

Fig. 2: Graphical Representation of Code Mixing

(iii) **Dead Code Insertion**: This method inserts null code inside the source code. It breaks decompilers to makes reverse-engineered code harder to analyze and extract the original form of code.

(iv) **The Pseudo code for the Dead code insertion:**

- Let us assume the L and U are Lower Bound and Upper Bound L=1 and U=SIZE
  MID = (L+U) / 2
- Insert the dead code at MID Position
  Chunks [MID] = Dead Code
- Adjust the Upper Bound of chunk array
  U=MID-1
- Inset the dead code on First partition of the code
  MID = (L+U) / 2
- Adjust the Lower Bound of chunk array
  L= MID+1
- Inset the dead code on Second partition of the code
  MID = (L+U) / 2
- End

Fig..3: Graphical Representation of Dead Code Insertion

(v) Lossless Compression: Run Length Encoding is a loss less compression technique which runs on the basis of the occurrence of data and replace it with single value and count[9].

Algorithm for RLE

Step 1 set the count value to zero
Loop: count = 0
Step 2 REPEAT the Steps and get next symbol
Step 3 Increment the count value UNTIL (symbol unequal to next one)
count = count + 1
Step 4 IF count > 1
Step 5 output count
Step 6 REPEAT FROM STEP 1 GOTO Loop

(vi) Cryptographic hash SHA 3: cryptographic hash function generates the message digest in which data is first absorbed into sponge and then result is squeezed. In the absorbing phase, message blocks are XORed into a subset of the state, which is then transformed as a whole using a permutation function *f*. In the "squeeze" phase, output blocks are read from the same subset of the state, alternated with the state transformation function *f*. The size of the part of the state that is written and read is called the "rate" (denoted *r*), and the size of the part that is untouched by input/output is called the "capacity" (denoted *c*). The capacity determines the security of the scheme. The maximum security level is half the capacity. Given an input bit string N, a padding function pad, a permutation function f that operates on bit blocks of width b, a rate r and an output length d, we have capacity $c = b - r$ and the sponge construction Z = sponge[f,pad,r] (N,d), yielding a bit string Z of length d, works as follows[10]:

- pad the input N using the pad function, yielding a padded bit string P with a length divisible by r (such that n = len (P)/r is integer)
- break P into n consecutive r-bit pieces P_0, ..., P_{n-1}
- initialize the state S to a string of b zero bits
- absorb the input into the state: for each block P_i:
- extend P_i at the end by a string of c zero bits, yielding one of length b
- XOR that with S
- apply the block permutation f to the result, yielding a new state S
- initialize Z to be the empty string
- while the length of Z is less than d:
- append the first r bits of S to Z
- if Z is still less than d bits long, apply f to S, yielding a new state S
- truncate Z to d bits

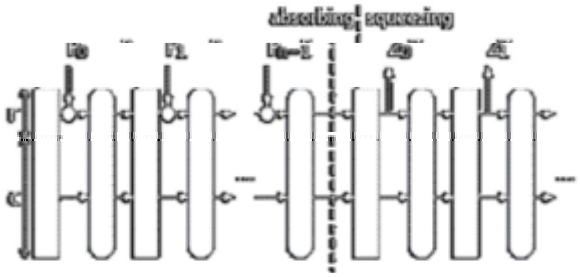

Fig..4: Function of SHA 3

(b) Data Design-In this section; we describe the methods used to evaluate the performance of this framework. We also identify the performance evaluation metrics that are identified for evaluation of the framework.

(i) Evaluation Metrics-We identify application execution time and consumed energy as two metrics of evaluation that are presented in Table 1 and explained as follows. These two metrics help us to obtain our aim and objectives in this study. Consumed energy and execution time are the most important established metrics to evaluate the lightweight properties of a typical MCC framework.

- **Battery Consumption**: Total battery consumption will be calculated using storing battery power values on session variables. Find the difference between Current state of battery level with last state of battery level
- **Execution Time**: Execution time is the amount of time taken to complete the entire life cycle of one prototype MA for one workload which is measured and stated in millisecond (ms). Different methods of generating execution time exist including manual and automatic data collection [8]. Similar to the energy, in order to avoid man made mistake and to obtain accuracy.

$$T_{total} = T_{mobile} + T_{rt} + T_{cloud}$$

Where T_{mobile}: - The time taken in local Machine, T_{rt}:-The time taken in delay (Round Trip) in network, T_{cloud}:-The times measure the computer latency of cloud computation.

Table 1
Unit and Symbols of Execution Time and Consumed Energy

Metric	Definition	Unit
Execution Time	Total Time takes to complete one task	nano Sec.
Consumed Energy	Total amount of energy consumed on the device to complete one task	Joule

VII RESULT AND DISSCUSSION

This section discusses experimentation findings of evaluating proposed by employing the prototype application. It presents analysis of Total Execution Time and Battery Consumption Cost of the mobile device for different web based application from the perspective of local and remote execution via Agent. In this experiment, we compare the finding values of Total Execution Time and Battery Consumption Cost with three different scenarios within same framework 1) Application runs under local machine, 2) Application runs on cloud without security parameters, 3) Application runs on cloud with three levels of security parameters. Results of performance evaluation generated via offload the five different web based applications are presented in this section in two parts. In the first part, data related to Execution Time Analysis is presented followed by Battery Consumed Energy in part .Experimental finding values are shown in Table 2 and Table 3 and Comparison of results are shown in Figure 4 and 5.

Table 2
Execution Time Analysis of Web Based Application

Total Execution Time in nano-second				
No. of Work load	Type of Mobile Application	Execution in Local Machine	Normal Offloading Execution	Secure Offloading Execution on Cloud
1	Google Home Page	125	180	190
2	Facebook Signup	171	202	214
3	Google Search	190	287	294
4	Song Online	310	402	418
5	Navigation	100	265	280

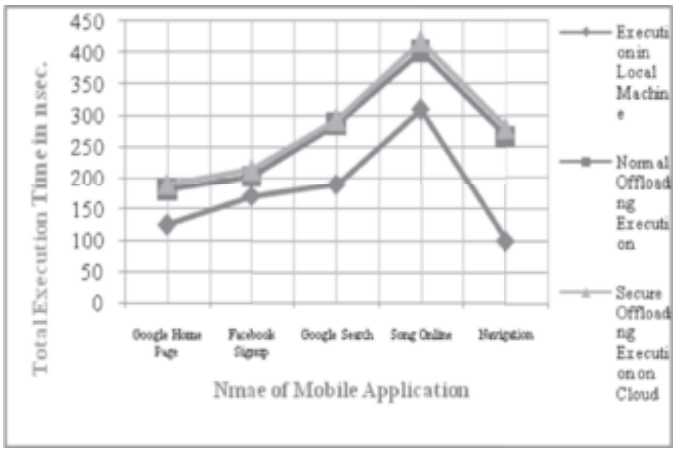

Fig. 4 Comparison of Total Execution Time

Table 3
Battery Consumption Analysis of Web Based Application

Total Battery Consumed in Joule				
No. of Work load	Type of Mobile Application	Execution in Local Machine	Normal Offloading Execution	Secure Offloading Execution on Cloud
1	Google Home Page	5	4.1	4.4
2	Facebook Signup	6.12	5.2	5.7
3	Google Search	5.99	3.72	3.9
4	Song Online	7.33	5.91	6.1
5	Navigation	18.2	16.77	16.9

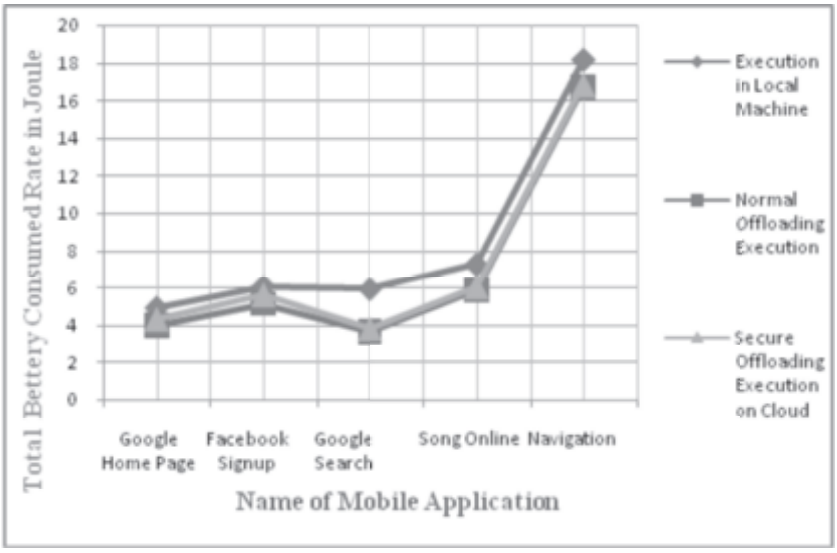

Fig. 4 Comparison of Total Battery Consumption

VIII CONCLUSION

In this work we have implemented energy efficient and secure framework for MCC so that user can trust on mobile cloud computing and enjoy the outcome without any worry. Here we have discussed the different types of obfuscation method and selected two methods to enhance security and privacy requirement, threats, concerns issues, risk associated to cloud and provide multiple securities in framework which fulfil complete cloud security requirements. As an experimental outcome, the secure framework affects the size of task, execution time, and reduces battery consumption. Finally it improves the overall function of the computation offloading using mobile.

REFERENCES

[1] Vasudevan M. (2001) ,"An Architecture of Class Loader System in Java Bytecode Obfuscatio", Indian Journal of Science and Technology, Vol 8(S2),PP. 291–294.

[2] Cachin, C., Keidar I., and Shraer.(2009) A. Trusting the cloud. ACM SIGACT News,PP. 81-86.

[3] Popovic K. & Hocenski, Z.(2010) "Cloud computing security issues and challenges", of the 33rd International Convention, PP .344-349.

[4] Grobauer, B., Walloschek, T., Stocker, E.(2011) "Understanding Cloud Computing Vulnerabilities", Security & Privacy, IEEE, Vol. 9, No. 2, PP.50-57.